Effizientes Reinforcement-Learning für autonome Navigation

Von der Fakultät für Ingenieurwissenschaften,
Abteilung Informatik und Angewandte Kognitionswissenschaft
der Universität Duisburg-Essen
zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von

Anastasia Noglik

aus

Uljanowsk (Russland)

Referent: Prof. Dr. Josef Pauli
Korreferent: Prof. Dr. Christian Igel
Tag der mündlichen Prüfung: 21. März 2012

Meinen Großeltern

Vorwort

Die vorliegende Arbeit entstand am Lehrstuhl Intelligente Systeme der Universität Duisburg-Essen unter der Leitung von Herrn Prof. Dr. (habil.) Josef Pauli. Ihm gilt mein ganz besonderer Dank für die mir zugestandene Freiheit in meiner Arbeit, eine exzellente Koordination des Projekts, die Übernahme des Referates, ergiebige Diskussionen und viele Anregungen, die mich stets weitergebracht haben. Herrn Prof. Dr. (habil.) Christian Igel vom Department of Computer Science an der Universität Kopenhagen gebührt mein Dank für die Übernahme des Korreferates, konstruktive Kritik und wichtige Vorschläge.

Frau Dr. Maria Sagrebin-Mitzel danke ich für eine unvergessliche Zeit, stundenlang verbrachte Diskussionen und moralische Unterstützung. Herrn Johannes Herwig möchte ich für die Geduld beim gemeinsamen Kampf mit dem Textverarbeitungsprogramm LaTex ganz herzlich danken. Herrn Michael Müller gilt mein Dank für viele hilfreiche Diskussionen und die Hilfe beim Robotersystem. Immer, wenn es nicht funktionieren wollte, hat er das Robotersystem wieder einsatzfähig gemacht. Meine Verbundenheit gilt Herrn Zhenyu Tang und den Doktoranden Zhao Kun, Jens Hoefinghoff, Michael Korn und Holger Wichert. Allen Kollegen, Diplomanden, Projektmitarbeitern, die am Lehrstuhl an der Entwicklung kognitiver Systeme gearbeitet haben, möchte ich für die angenehme Arbeitsatmosphäre danken.

Besonderer Dank gilt allen Diplomanden, die ich während meiner Zeit am Lehrstuhl betreuen durfte, unter anderem Frau Anne Wolter und den Herren Thomas Köpsel, Sebastian Papierok, Andreas Mai, Kevin Schewior, Mikhail Zhigun und vielen weiteren. Herrn Thomas Köpsel möchte ich für die Entwicklung realitätsnaher Simulation danken. Herrn Sebastian Papierok möchte ich meinen Dank sagen für die Entwicklung der Simulationsumgebung für den Scorpion-Roboter und die ersten Realisierungen des Reinforcement-Learning-Algorithmus. Bei Herrn Andreas Mai möchte ich mich besonders für die enge und erfolgreiche Zusammenarbeit bei der Entwicklung der DPA-RL-Architektur bedanken.

Herrn Thomas Günther danke ich für die Unterstützung, jederzeit war er da, wenn etwas mit der Hard- oder Software nicht stimmte. Mein Dank gilt Frau Marianne Appelt und Frau Marion Handke, die mich bei organisatorischen Angelegenheiten immer unterstützt haben.

Meiner Familie, insbesondere meinen Eltern und Großeltern, die mich immer unterstützt und angetrieben haben, und meinem Bruder, auf den ich immer zählen kann, bin ich für den familiären Rückhalt dankbar. Meinem Mann Adam danke ich für die Motivation und vor allem für die Geduld und Unterstützung.

Anastasia Noglik

Dortmund, im November 2011

Zusammenfassung

Die vorliegende Arbeit mit dem Titel ”Effizientes Reinforcement-Learning für autonome Navigation” ist der Entwicklung autonomer rationaler Agenten in dynamischen Umgebungen gewidmet. Der Begriff Lernen ist zentral für die vorliegende Arbeit.
Rationale Agenten stellen nach Meinung von zahlreichen Autoren einen Schlüssel zur künstlichen Intelligenz dar. Ein Agent ist dann rational, wenn er in jedem Schritt eine Aktion ausführt, die den maximalen Gesamtnutzen für die aktuelle Situation verspricht. Der Nutzen wird in Abhängigkeit von den verfolgten Zielen definiert. Die Verteilung des Gesamtnutzens über den Zustandsraum des Agenten wird in der Nutzenfunktion zusammengefasst. Die angestrebte Lernfähigkeit des Agenten wird über eine Anpassung der Nutzenfunktion an die optimale Nutzenfunktion, die die optimale Lösung für das vorgegebene Problem darstellt, erzielt. Das Lernen wird mit einer Reinforcement-Learning-Methode realisiert. Ein großer Nachteil dieser Methode ist ihre langsame Lerngeschwindigkeit.
Ein effizienter Lernvorgang des rationalen Agenten ist das erste angestrebte Ziel der vorliegenden Arbeit. Das zweite verfolgte Ziel liegt in der Entwicklung und Erprobung von rationalen Agenten für anspruchsvolle Navigationsprobleme. Dabei werden konkrete Realisierungen der autonomen Robotersysteme konstruiert, ihre Stichhaltigkeit bestätigt und die Qualität der erzielten Lösungen kontrolliert. Die beiden verfolgten Ziele bilden den Kern der vorliegenden Arbeit.
Um einen effizienteren Lernvorgang zu erzielen werden viele Bestandteile der rationalen Agenten modifiziert. Der Lernmechanismus wird um heuristische Funktionen erweitert. Ein weiterer Schritt ist die Modifikation des Aktionsraumes.
Mithilfe der Einbindung einer heuristischen Funktion in den Lernprozess (SARSA*-Algorithmus) bzw. bei der Entscheidung über die nächste Aktion (ϵ^*-gierige Strategie) wird die Lerngeschwindigkeit des Systems in den meisten Fällen positiv beeinflusst. Das beste Ergebnis wird für das einfache Gitterweltproblem mit einer endlichen Anzahl an Zuständen und Aktionen erzielt. Dieses Ergebnis weist auf das große Potenzial der eingeführten Erweiterungen des Lernmechanismus hin. Für den SARSA*-Algorithmus werden 50% weniger Lernschritte und bei Anwendung der ϵ^*-gierigen Strategie 12.5% weniger Lernschritte bis zum Erreichen der optimalen Strategie im Vergleich zum gewöhnlichen SARSA-Algorithmus benötigt. In komlexeren Problemarten ist die erreichte Beschleunigung des Lernprozesses weniger signifikant. Für das Mountain-Car-Problem, ein Benchmarkproblem für Reinforcement-Learning-Algorithmen, wird eine Beschleunigung des Lernprozesses von 10% bei Verwendung des SARSA*-Algorithmus und von bis zu 5% bei Anwendung der ϵ^*-gierigen Strategie erreicht. Im Navigationsproblem wird eine starke Abhängigkeit der Auswirkung der heuristischen Funktion vom Aufbau der Szenarien beobachtet. Der Mittelwert für die Beschleunigung des Lernprozesses für drei unterschiedliche Szenarien liegt bei Anwendung des SARSA*-Algorithmus bei 16% und bei Anwendung der ϵ^*-gierigen Strategie in zwei Szenarien bei 21%. Die vorgeschlagenen Erweiterungen erfordern nur wenige zusätzliche Rechenoperationen.
Die Gestaltung des Aktionsraumes ist stark mit der verwendeten Architektur des Agenten verbunden und erlaubt eine weitere Möglichkeit den Lernprozess positiv zu beeinflussen. Zuerst wird der auf dem niedrigen Abstraktionsniveau durchgeführte Lernprozess untersucht. Der Aktionsraum besteht aus Aktionen, die durch diskrete Steuerungsbefehle repräsentiert werden. Diese Agentenart wird um eine reak-

tive Fähigkeit erweitert. Die erweiterte Agentenart wird als autonomer Reinforcement-Learning-Agent (autonomer RL-Agent) bezeichnet. Der Vorinitialisierungsschritt bietet die Möglichkeit die vorhandenen, im Lernprozess nicht angewendeten Informationen in die Nutzenfunktion einzubinden. Durch den Vorinitialisierungsschritt, die Einbindung des Reflexes und der heuristischen Funktion im autonomen RL-Agenten wird zwar eine Effizienzsteigerung des Lernprozesses erreicht. Es werden aber sehr viele Episoden und Lernschritte benötigt, da der Parametervektor im Raum höherer Dimensionalität liegt, wobei eine Episode aus der Aktionskette besteht, die den Agenten vom Start bis zum Ziel führt.
Anschließend wird der Lernprozess auf einem höheren Abstraktionsniveau aufgebaut und untersucht. Der Doppelpass-Architektur-Reinforcement-Learning-Agent (DPA-RL-Agent) stellt das in der vorliegenden Arbeit endgültige Robotersystem zur Lösung von Navigationsproblemen dar. Der DPA-RL-Agent beinhaltet eine Doppelpass-Architektur (DPA) und lernt auf Grundlage von abstrakten Aktionen. Die spärliche Darstellung des Zustandsraumes, der aus wichtigen Entscheidungsknoten besteht, trägt entscheidend zum schnellen und effizienten Lernvorgang bei. Der vorgeschlagene Ansatz der DPA-RL-Agenten wird in zahlreichen Anwendungen und unterschiedlichen Szenarien untersucht. Der Lernvorgang findet direkt (ohne Simulationsphase) in den komplexen realen Szenarien, ohne Wissen über den Aufbau der Umgebung statt. Nur wenige Episoden werden zum Erlernen einer Trajektorie benötigt, die der optimalen Trajektorie nahe kommt. Der Ansatz ist modular, flexibel und kann für unterschiedliche Konfigurationen der Hardware des Roboters angewendet werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

”Symbolverarbeitung oder nicht, diese Frage legt die neue Riege der Forscher erst mal beiseite und setzt dafür eine neue Prämisse: Intelligenz ist keine einheitliche, unteilbare Fähigkeit, sondern sie stellt die Summe vieler Einzelfähigkeiten dar. Die vermeintliche Komplexität eines Verhaltens ist über einen längeren Zeitraum gesehen zum großen Teil eine Reflektion der Umwelt, in der es sich bewegt. Denn ob Ameise oder Mensch - ein sich verhaltendes System kann recht simpel wirken. Nicht Verhalten an sich, sondern Verhalten im Kontext einer gegebenen Umwelt bestimmt Intelligenz.” [Auf dem Hövel, 2002]. In einem Buch von Russell und Norvig wird die künstliche Intelligenz (KI) ”als die Betrachtung von Agenten, die Wahrnehmungen aus der Umwelt erhalten und Aktionen ausführen” definiert. Immer mehr Autoren betrachten das Konzept der rationalen Agenten als zentral für den Zugang zur künstlichen Intelligenz. ”Jeder dieser Agenten implementiert eine Funktion, die Wahrnehmungsfolgen auf Aktionen abbildet, und wir beschreiben unterschiedliche Möglichkeiten, diese Funktionen darzustellen, wie z. B. ... reaktive Agenten, Echtzeitplaner, neuronale Netzwerke und entscheidungstheoretische Systeme.” [Russell & Norvig, 2004, Vorwort, S. 10]

1.1 Ziele und Motivation

Als Motivation für diese Arbeit diente sowohl die globale Absicht einen Beitrag auf dem Gebiet der künstlichen Intelligenz zu leisten, aber auch der lokale Wunsch ein effizientes autonomes Robotersystem zu entwickeln, das die Fähigkeit zum Lernen besitzt und in einer Umgebung mit Menschen ohne Sicherheitsbedenken einsetzbar ist.
Das erste Ziel der vorliegenden Arbeit liegt in einer effizienteren Gestaltung einer bereits bekannten Lernmethode. Dieses Ziel ist global und ist theoretischer Natur. Nach der Lösung dieses Ziels kommt das zweite in der vorliegenden Arbeit verfolgte Ziel zur Geltung und liegt in der Entwicklung eines konkreten rationalen Agenten, der die Eigenschaften und Gegebenheiten realer Roboter beinhaltet und ein Navigationsproblem erfolgreich löst. Ein *rationaler Agent* soll entwickelt werden, der unter realen Bedingungen sein Wissen vervollständigen kann, die aktuelle Situation verarbeiten kann und in Hinsicht auf vordefinierte Ziele und gewonnenes Wissen rational handeln kann. In vielen Fällen liegt das Problem vor, dass gar kein oder nur ein unvollständiges Umgebungsmodell existiert. Dynamische Veränderungen in der Umgebung können nicht ausgeschlossen werden. Z. B. können auf dem Weg des Agenten plötzlich Hindernisse auftauchen. Die Fähigkeit im Onlinebetrieb (Echtzeit-KI) zu lernen ist dadurch eine sehr wichtige Eigenschaft eines rationalen Agenten. Wissenschaftliche Arbeiten auf diesem Gebiet sind sehr wichtige Bestandteile der Theorie des rationalen Agenten [Russell & Norvig, 2004, Kap. 27.2, S. 1179]. Zum Lösen dieses Ziels werden wesentliche Bereiche der künstlichen Intelligenz angewendet: *Wissen*, *Schließen*, *Planen*, *Lernen*, Methoden wie ein rationaler Agent seine Umgebung wahrnimmt - *Wahrnehmung* und die Möglichkeit wie der Agent Pläne in Aktionen umsetzt - *Agieren*. Um diese Ziele zu erreichen, werden hier auch bereits existierende Verfahren kombiniert, die sich die Ideen und Architektu-

ren aus unterschiedlichen Gebieten der künstlichen Intelligenz zunutze machen. Durch die Entwicklung rationaler Agenten, die effizient und schnell ein vorgegebenes Navigationsproblem im Onlinebetrieb[1] lösen, werden beide Ziele erreicht.

1.2 Problemstellung

Die verfolgten Ziele rufen zwei wichtige Probleme hervor. Zuerst wird die theoretische Frage gestellt, wie und unter welchen Rahmenbedingungen die schon vorhandene Lernmethode effizienter gestaltet werden kann. Das zweite Problem ist durch die konkrete Anwendung der vorhandenen und entwickelten Methoden im vorgegebenen Robotersystem geprägt. Dabei wird die Gestaltung des autonomen Robotersystems, das die effiziente Lernmethode beinhaltet, in den Vordergrund gestellt.
Die in der vorliegenden Arbeit vorgeschlagenen effizienten Lernmethoden werden für unterschiedliche Problemarten, die durch unterschiedliche rationale Agenten gelöst werden, angewendet. Wenn die Erweiterungen unabhängig von der konkreten Realisierung des rationalen Agenten in unterschiedlichen Problemarten anwendbar sind und in unterschiedlichen Problemarten eine Steigerung der Effizienz erzielen, weisen die effizienten Lernmethoden das gewünschte Verhalten auf.
Eine praxisorientierte Entwicklung und Anwendung von effizienten Lernverfahren für Navigationsprobleme, also die Konstruktion eines rationalen Agenten, der sich an seine Umgebung anpasst und das Navigationsproblem erfolgreich löst, bereitet die Lösung für das zweite in der vorliegenden Arbeit verfolgte Problem vor. Der rationale Agent wird durch eine Realisierung des Leistungselements sowie eine Vorgabe der Sensoren und Aktuatoren konstruiert.

1.3 Aufbau der Arbeit

In Abbildung 1.1 ist die Aufteilung der vorliegenden Arbeit schematisch dargestellt. Im ersten Teil der Arbeit steht das Lernelement im Vordergrund, siehe rotes Rechteck. Im zweiten Teil der Arbeit wird ein besonderes Augenmerk auf die Entwicklung des rationalen Agenten zur Lösung eines Navigationsproblems gelegt, siehe durch gestrichelte Linie gekennzeichnetes Rechteck.
Die grundlegenden Begriffe und Lernalgorithmen werden in Kapitel 2 und 3 eingeführt. Ein Vorschlag, wie der SARSA-Algorithmus als eine bekannte Lernmethode effizienter gestaltet werden kann, wird in Kapitel 4 eingeführt. Dieses Kapitel beinhaltet sowohl theoretische als auch experimentelle Ergebnisse.
Der zweite Teil der vorliegenden Arbeit wird der Entwicklung eines rationalen Agenten zur Lösung eines Navigationsproblems gewidmet. Dieser Teil beinhaltet die Kapitel 5 bis 8. Die entwickelten Realisierungen stellen die autonomen Robotersysteme dar, die fähig sind den optimalen Weg im Navigationsproblem zu erlernen. Zwei Arten der möglichen Realisierungen des rationalen Agenten werden eingeführt. Beide Realisierungen beinhalten den gleichen Lernalgorithmus, der im ersten Teil dieser Arbeit eingeführt wird. Die im ersten Teil der Arbeit eingeführten Vorschläge zur Erweiterung des Lernelements werden im zweiten Teil der Arbeit mithilfe der entwickelten Agententypen, die Navigationsprobleme lösen, untersucht.
Der in Kapitel 5 betrachtete Reinforcement-Learning-Agent (RL-Agent) beinhaltet den SARSA-Algorithmus und ist auf die Lösung der Navigationsaufgabe ausgerichtet. Diese Agentenart operiert mit Ak-

[1] Der Begriff online wird hier in dem Sinne gebraucht, dass der Agent nicht auf einen vollständigen Eingabedatensatz wartet, sondern die ankommenden Daten sofort verarbeitet werden.

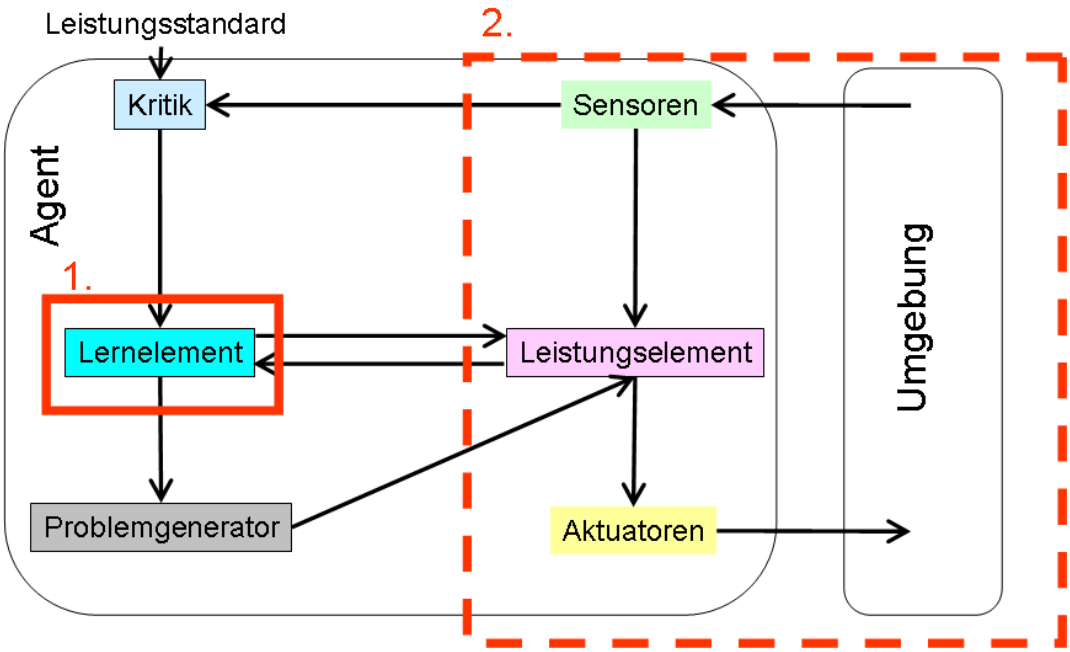


Abbildung 1.1: Lernender Agent nach Russell [Russell & Norvig, 2004, Kap. 2.4.6, S. 80]. Schematische Darstellung der zwei Schwerpunkte der vorliegenden Arbeit, rote durchgezogene Linie und rote gestrichelte Linie

tionen niedrigen Abstraktionsgrades, die keine Autonomie aufweisen und in jedem Zeitschritt eine Entscheidung des Leistungselements benötigen.
In Kapitel 6 werden mögliche abstrakte Aktionen sowie die Methoden zum Erzeugen solcher abstrakter Aktionen dargestellt. Die abstrakten Aktionen repräsentieren Reflexe, die autonom und ohne Zugriff auf Entscheidungen des Leistungselements ausgeführt werden.
Die nächste Agentenart verwaltet die abstrakten Aktionen und erlernt mithilfe dieser Aktionen das gewünschte Verhalten. Diese Agentenart wird in Kapitel 7 eingeführt. Das Lernen auf Grundlage der abstrakten Aktionen bringt gewisse Schwierigkeiten mit sich, die durch eine im Bereich der Fußball-Roboter bekannte Doppelpass-Architektur (DPA) beseitigt werden können. Der entwickelte Doppelpass-Architektur-Reinforcement-Learning-Agent (DPA-RL-Agent) besitzt die DPA-Architektur und ist um die Fähigkeit zum Lernen erweitert.
In Kapitel 8 werden zahlreiche, mit dem DPA-RL-Agenten gewonnene Ergebnisse dargestellt. Viele Untersuchungen werden unter Echtzeit-Bedingungen in der Realität durchgeführt.

1.4 Wissenschaftlicher Beitrag

Für beide verfolgten Ziele der vorliegenden Arbeit liegt ein wissenschaftlicher Beitrag vor. Die vorgeschlagenen heuristischen Erweiterungen für den RL-Algorithmus sind grundlegende Ideen. Diese Erweiterungen können unabhängig von der Problemstellung angewendet werden und führen in den meisten Fällen zu einem effizienteren Lernvorgang. Die ersten Ideen sind von der Autorin bereits in [Noglik et al., 2009] präsentiert worden. Die vorliegende Arbeit beinhaltet die theoretische Basis, in welchem Rahmen die vorgeschlagenen Erweiterungen nie schädlich sein können. Zwei wichtige Säulen der aufgebauten Theorie, die fundierte theoretische Herleitung und die praxisorientierte Bestätigung der theoretisch erzielten Ergebnisse, bilden ein stabiles Fundament für die vorgeschlagenen heuristischen Erweiterungen

der Reinforcement-Learning-Methode. Mithilfe zahlreicher Experimente für unterschiedliche Problemarten wird die Qualität der vorgeschlagenen Erweiterung in der vorliegenden Arbeit bestimmt.
Über die Gestaltung des Aktionsraumes wird ein weiterer Schritt zur Effizienzsteigerung der Lernmethode des rationalen Agenten gemacht. Zunächst wird der aus elementaren diskreten Aktionen bestehende Aktionsraum um Reflexe in Gefahrensituationen erweitert. Diese Realisierung des rationalen Agenten wird als Reinforcement-Learning-Agent (RL-Agent) bezeichnet. Die ersten Überlegungen, wie der RL-Agent aussehen kann, wurden von der Autorin in [Noglik & Pauli, 2009] dargestellt. Der in der vorliegenden Arbeit entwickelte RL-Agent beinhaltet den Lernalgorithmus, der die Reflexe im Lernvorgang mitverarbeitet. Die Übergabe der Steuerung in einer Gefahrensituation an die Reflexe hat Ähnlichkeiten zum Verhalten von Menschen. Für diese Agentenart wird der Vorinitialisierungsschritt eingeführt, der die Möglichkeit bietet Informationen über den Aufbau der Umgebung in den Lernprozess einzubinden. Der RL-Agent mit dem durchgeführten Vorinitialisierungsschritt stellt eine gute, adaptionsfähige Alternative zum Kraftfelder-Ansatz dar. Diese Agentenart kann ihre Nische in der autonomen Navigation finden.
Wenn der Aktionsraum aus Aktionen besteht, die reaktiv und autonom sind, wird der Lernprozess auf der höheren Abstraktionsebene angesiedelt. Diese Aktionen werden als abstrakte Aktionen bezeichnet. Die möglichen Realisierungen der in der vorliegenden Arbeit angewendeten abstrakten Aktionen sind in [Papierok et al., 2008] sowie [Köpsel et al., 2007] präsentiert worden. In der vorliegenden Arbeit wird eine weitere Möglichkeit zum Erzielen der abstrakten Aktionen beschrieben und mit den anderen Realisierungen verglichen. Die Anwendung der Aktionen höheren Abstraktionsgrades erfordert eine komplexe Architektur. In der vorliegenden Arbeit wird die existierende Doppelpass-Architektur angewendet. Der DPA-RL-Agent lernt auf Grundlage der abstrakten Aktionen. In der vorliegenden Arbeit wird dem Lernen auf höherer Abstraktionsebene besondere Aufmerksamkeit gewidmet. Die unterschiedlichen Realisierungen der abstrakten Aktionen, sowie die Auswirkung der heuristischen Erweiterung werden untersucht. Der DPA-RL-Agent kann unter anderem im Bereich der Fußballrobotik, Servicerobotik oder auch Rettungsrobotik seine Anwendung finden. Das Grundgerüst der DPA-RL-Agenten ist sehr modular und flexibel, so dass diese Agentenart für Aufgaben angewendet werden kann, in denen Entscheidungsketten auf höheren Abstraktionsebenen erlernt werden sollen.

Kapitel 2

Rationale und lernende Agenten

”Computerintelligenz ist die Studie des Entwurfs intelligenter Agenten” [Poole et al., 1998], so lautet die Definition der modernen Intelligenz nach Poole. Intelligenz wird in [Pfeifer & Bongard, 2006] als Wechselwirkung zwischen physikalischen und informationstechnischen Prozessen verstanden und auch als verkörperte Intelligenz (engl.: embodied intelligence) bezeichnet. Und erst über Interaktion des Körpers des Agenten mit der Umgebung werden die benötigten, intelligenten Handlungszüge erzielt. An dieser Stelle wird die Intelligenz auch als Rationalität bezeichnet, da zum Beispiel Russell und Norvig künstliche Intelligenz über den Entwurf eines rationalen Agenten definieren. Die Definition künstlicher Intelligenz über den Entwurf eines rationalen Agenten beinhaltet zwei wichtige Vorteile. Erstens ist der auf dem Entwurf rationaler Agenten basierende Ansatz allgemeiner als der Ansatz der ”Denkregeln”, weil ”Denkregeln” nur eine von mehreren Möglichkeiten zum Erzielen von Rationalität darstellen. Zweitens ist es ”für den wissenschaftlichen Fortschritt sinnvoller” diesen Ansatz anzuwenden als Ansätze, die auf ”menschlichen Gedanken basieren”, da der Begriff Rationalität klar definiert ist [Russell & Norvig, 2004, Kap. 1.1.4, S. 22].
In diesem Abschnitt werden die grundlegenden Begriffe des abstrakten Konzepts der rationalen Agenten sowie die problembezogenen, konkreten Realisierungen eingeführt.
In Kapitel 2.1 werden zuerst wichtige Begriffe wie *Agent*, *Rationalität*, *Lernen* und *rationaler Agent* erläutert. In Kapitel 2.2 wird dann basierend auf dem rationalen Agenten ein weiterer wichtiger Agententyp, der *lernbasierte Agent*, eingeführt. Beide Agententypen sind zuerst abstrakte Konstruktionen.
Die konkrete Realisierung des Agenten ist durch die Problemstellung sowie die Gegebenheiten der Hardware des Agenten geprägt. Die für die vorliegende Arbeit wichtigen Problemstellungen und die mit gegebener Hardware verbundenen Architekturen des Agenten werden im vorliegenden Kapitel eingeführt. Das Mountain-Car-Problem (MCP) sowie die Architektur des MCP, die zur Lösung dieses Problems angewendet wird, werden in Kapitel 2.3 eingeführt. Das zentrale Problem in dieser Arbeit und in der mobilen Robotik ist das Navigationsproblem. Dieses wird als Benchmark für die Prüfung der Tauglichkeit der erzielten konkreten Realisierung des rationalen Agenten ausgewählt. Das konkrete Navigationsproblem mit den Randbedingungen, die für die vorliegende Arbeit gelten, wird in Kapitel 2.4 eingeführt. Die Architektur des Agenten, der auf die Lösung des Navigationsproblems ausgerichtet ist, wird in Kapitel 2.5 präsentiert.
In Kapitel 2.6 wird das Konzept der realitätsnahen Simulation eingeführt. Mit diesem Konzept konnten weichere Grenzen zwischen Simulation und Realität geschaffen werden, so dass kein Nachbearbeitungsschritt beim Umstieg von der Simulation in reale Umgebungen benötigt wird.
Abschließend wird in Kapitel 2.7 eine Übersicht über die existierenden Arten der Architekturen in der mobilen Robotik gegeben.

2.1 Rationale Agenten

2.1.1 Agent

Es gibt einen abstrakten Begriff *Agent* und daneben die konkreten Realisierungen dieses Begriffs, die durch die Aufgabenstellung und die Gegebenheiten der Hardware geprägt sind [Russell & Norvig, 2004, Kap. 2.4, S. 70]. ”Ein Agent ist alles, was seine Umgebung über Sensoren wahrnehmen kann und in dieser Umgebung durch Aktuatoren handelt” [Russell & Norvig, 2004, Kap. 2.1, S. 55]. Die Struktur des Agenten besteht aus seiner Architektur und dem Agentenprogramm. Das Programm implementiert die Agentenfunktion, die die Wahrnehmungen auf Aktionen abbildet. Die Architektur beinhaltet einen Computer sowie physische Sensoren und Aktuatoren des Agenten und ist für das Entgegennehmen der Wahrnehmungen von den verfügbaren Sensoren, das Ausführen des Programms und das Weitergeben ausgewählter Aktionen an die Aktuatoren verantwortlich.
Zur Bestimmung der geforderten Intelligenz wird der Begriff *Rationalität* ins Spiel gebracht und anschließend zur *Nutzenfunktion* überführt. Rationalität wird in Abhängigkeit von den verfolgten Zielen durch die Messung des Erfolgs definiert. Der Agent ist erfolgreich, wenn er das Ziel erreicht.
Rationale Agenten können unterschiedliche Strukturen aufweisen. Zum Beispiel hat ein Roboter andere Bestandteile als eine Suchmaschine im Internet, beide sind aber intelligente Agenten mit verschiedenen Konfigurationen aus Aktuatoren und Sensoren. In Abhängigkeit von den Gegebenheiten eines Agenten sollen die möglichen Aktionen definiert werden. Die Aktionen selbst können unterschiedliche Abstraktionsgrade aufweisen. Die Verwaltung dieser Aktionen wird in Abhängigkeit von der Definition der vorhandenen Aktionen implementiert.

2.1.2 Rationalität und Nutzenfunktion

Zwischen Rationalität und Allwissenheit soll streng unterschieden werden. Rationalität darf nicht mit Perfektion verwechselt werden, die Allwissenheit voraussetzt. Ein intelligenter Agent muss rational handeln können, was keine Allwissenheit erfordert. Mit rationalem Handeln sind zwei Prozesse verbunden, eine umfassende Analyse einer Situation vor dem Handeln und ein schnelles Handeln. Diese Prozesse stehen oft in Konkurrenz zueinander. Das Sammeln von Informationen und die Exploration der Umgebung sind wichtige Bestandteile der Rationalität.
”Ein rationaler Agent soll für jede mögliche Wahrnehmungsfolge eine Aktion auswählen, von der erwartet werden kann, dass sie seine *Leistungsbewertung*[2] maximiert, wenn man seine Wahrnehmungsfolge sowie vorhandenes Wissen, über das er verfügt, in Betracht zieht.” [Russell & Norvig, 2004, Kap. 2.2.2, S. 59].
Jeder rationale Agent muss sich so verhalten, ”als besäße er eine Nutzenfunktion, deren erwarteten Wert er zu maximieren versucht” [Russell & Norvig, 2004, Kap. 2.4.5, S.78], so wird der Zusammenhang zwischen der Intelligenz, der Rationalität und dem Optimierungsproblem über die Nutzenfunktion hergestellt.
”Eine Nutzenfunktion bildet einen Zustand oder eine Zustandsfolge auf eine reelle Zahl ab, die den zugehörigen Grad der Zufriedenheit beschreibt”, so dass die Ziele, die zum Beispiel in Konflikt zueinander stehen können, mithilfe der Nutzenfunktion gegeneinander abgewogen werden. Die Nutzenfunktion unterscheidet sich von den vorgegebenen Zielen dadurch, dass die Ziele eine grobe binäre Unterscheidung zwischen den Zuständen darstellen: ”glücklich”, wenn das Ziel erreicht wird, oder ”unglücklich”, wenn

[2] Die Leistungsbewertung kann mit einer Nutzenfunktion erfolgen.

das Ziel nicht erreicht wird. Eine allgemeine Leistungsbewertung sollte aber einen Vergleich verschiedener Zustände der Welt erlauben. Diese Bewertung berücksichtigt wie glücklich die Zustände den Agenten machen, wenn sie erzielt werden. Dabei kann der Begriff *glücklich* mit nützlich ersetzt werden [Russell & Norvig, 2004, Kap. 2.4.5, S. 78].
Die nächste Frage ist, wie die Nutzenfunktion erzielt werden kann. Zur Bestimmung bzw. zum Erlernen der Nutzenfunktion existieren zahlreiche Ansätze. Im nächsten Kapitel wird in diesem Zusammenhang der lernbasierte Agent, der Agent der die Fähigkeit zum Lernen besitzt, eingeführt. An dieser Stelle wird die Definition des Begriffs ”Lernen” aus der Psychologie und der künstlichen Intelligenz eingeführt.

2.1.3 Lernen

Neben der Rationalität zeichnet sich ein rationaler Agent durch seine Lernfähigkeit aus, also die Anwendung der Erfahrungen und der daraus resultierenden Fähigkeit zur selbständigen Verhaltensänderung. Der rationale Agent soll seine Erfahrungen sammeln, verändern und erweitern können [Russell & Norvig, 2004, Kap. 2.2.3, S. 61 ff].
Mehrere Quellen aus unterschiedlichen Wissenschaftsbereichen wurden zu Hilfe genommen, um den Begriff ”Lernen” an dieser Stelle zu spezifizieren. Dieser Begriff wird aus dem Gebiet der künstlichen Intelligenz im Bereich der Informatik übernommen, wobei die im Bereich der Psychologie verwendeten Definitionen dieses Begriffs auch berücksichtigt werden.
Lernen ist der Prozess zur Erstellung von Wissen, das für die entscheidungssuchende Komponente erforderlich ist. ”Das Lernen in intelligenten Agenten kann als Prozess der Veränderung der einzelnen Komponenten[3] des Agenten zusammengefasst werden, um die Komponenten besser mit der verfügbaren Feedback-Information abzustimmen und damit die Gesamtleistung des Agenten zu verbessern” [Russell & Norvig, 2004, Kap. 2, S. 82].
Als Lernfähigkeit wird in [Wikipedia, 2011c] die Eigenschaft eines Organismus Informationen speichern zu können und diese für eigene Zwecke zu nutzen bezeichnet.
Im Bereich der Psychologie wird der Begriff ”Lernen” als ”ein Veränderungsprozess”, ”der als Ergebnis individueller Erfahrungen auftritt” eingeführt. Lernforscher untersuchen den ”Prozess des Lernens”, als auch ”das Ergebnis, also jene langfristige Veränderung im Verhalten eines Individuums, die aus Lernerfahrungen resultieren”. ”Die Psychologie des Lernens umfasst sowohl den Erwerbsprozess als auch das daraus resultierende langfristige Verhalten” [Mazur, 2006, Kap. 1, S. 20 ff].
Zusammengefasst umfasst Lernen die Veränderungen, die das Gesamtverhalten des Individuums verbessern. Die Verbesserungen werden durch vordefinierte Ziele gemessen und mit dem Feedback der Umgebung verbunden.
Um die Lernfähigkeit messen zu können, werden an dieser Stelle ein paar Kriterien eingeführt, durch die die Fähigkeit des Agenten zum Lernen (Lernfähigkeit) erfasst werden kann. Diese Kriterien sind *Generalisierung* und *Adaptivität* bzw. *Flexibilität*.
Als Generalisierung (Verallgemeinerung) wird in der Verhaltensbiologie und Physiologie die Übertragung einer auf einen speziellen Reiz hin erlernten Reaktion auf ähnliche Reize (Konditionierung) verstanden [Mazur, 2006, Kap. 3.6.1, S. 87].
Anpassungsfähigkeit bzw. Adaptivität oder auch Flexibilität genannt sind die Fähigkeiten eines Lebewesens zur Veränderung oder Selbstorganisation, die Fähigkeit sich auf veränderte Anforderungen und Gegebenheiten einer Umwelt einstellen zu können, [Wikipedia, 2011a].

[3] Kritik, Lernelement, Problemgenerator, Leistungselement, siehe Abbildung 2.2

2.1.4 Zusammenhänge der entwickelten Agententypen

Die Architektur des Agenten wird in Abhängigkeit von den Gegebenheiten aus der Problemstellung definiert. In dieser Arbeit werden zwei grundsätzliche Problemarten betrachtet, das Mountain-Car-Problem [Sutton & Barto, 1998, Kap 8.4, S. 214 ff] und das Navigationsproblem [Stenzel, 2002]. Für die Lösung beider Problemarten werden zwei unterschiedliche Architekturen des Agenten entwickelt. Der MCP-Agent für die Lösung des Mountain-Car-Problems und der Roboter-Agent für die Lösung des Navigationsproblems. In der vorliegenden Arbeit wenden alle Agententypen den Reinforcement-Learning-Ansatz als Lernmechanismus an. Der Roboter-Agent wird wiederum in zwei Varianten realisiert. Die erste einfachere Variante ist der Reinforcement-Learning-Agent, der als Kern den Reinforcement-Learning-Ansatz beinhaltet. Die zweite mögliche Realisierung des Roboter-Agenten ist der Doppelpass-Architektur-Reinforcement-Learning-Agent (DPA-RL-Agent). Diese Agentenart beinhaltet als Kern die Doppelpass-Architektur, die um die Fähigkeit zum Lernen erweitert ist.
In der Abbildung 2.1 sind die Zusammenhänge der eingeführten Agentenarten schematisch dargestellt.

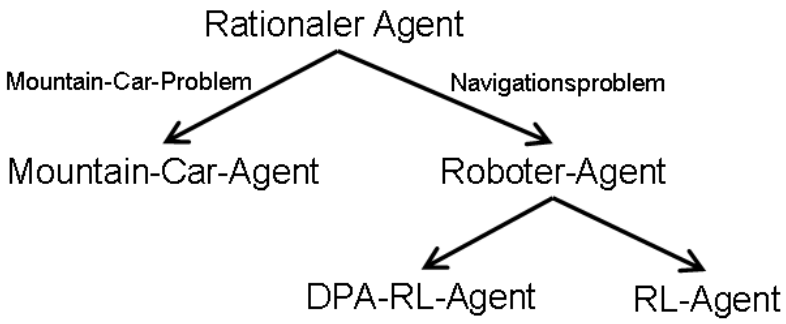


Abbildung 2.1: Übersicht über die in der vorliegenden Arbeit entwickelten Agententypen

2.2 Lernende Agenten

Die Hauptaufgabe rationaler Agenten ist eine Nutzenfunktion zu erlernen. Auf Grundlage der Nutzenfunktion werden rationale Handlungen des Agenten bestimmt. Das Lernelement der lernbasierten Agenten beinhaltet die Methode zur Ermittlung der gesuchten Nutzenfunktion.
Ein Agent, der rational handelt und die Fähigkeit zum Lernen besitzt, wird als lernender Agent bezeichnet. Ein Agent agiert durch Aktivierung seiner *Aktuatoren* in der Umgebung. Die Umgebung ”gibt” eine Antwort auf die durchgeführte Interaktion, die der Agent mithilfe seines Wahrnehmungsmodells (*Sensoren*) wahrnimmt.
Der lernende Agent besteht aus vier konzeptionellen Komponenten: Leistungselement, Lernelement, Kritik und Problemgenerator, siehe Abbildung 2.2.
Das *Leistungselement* nimmt Wahrnehmungen auf und entscheidet, welche Aktionen ausgeführt werden. Das Leistungselement beinhaltet eine Nutzenfunktion, die die Auswertung der möglichen Aktionen in der aktuellen Situation durchführt. Es kann als einfacher Reflex-Agent oder als komplexerer nutzenbasierter Agent realisiert werden. Der einfache Reflex-Agent stellt eine direkte Abbildung der Sensorinformationen auf Befehle an die Aktuatoren dar. Der komplexere, nutzenbasierte Agent verwaltet die internen Zustände des Agenten, baut ein Modell der Umgebung auf, passt es an und hat die Möglichkeit zu bewerten, wie gut die neu erreichten Zustände für den Agenten sind, um sein globales ”glücklich sein” zu erreichen. Für den Aufbau des internen Modells der Umgebung und für die Bestimmung des internen

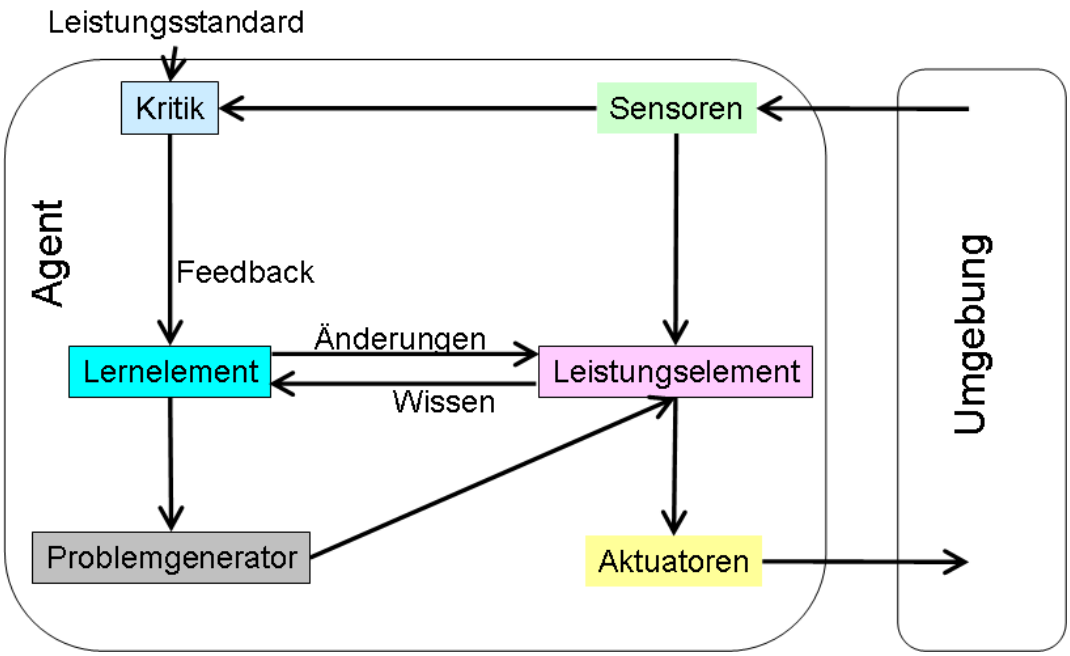


Abbildung 2.2: Lernender Agent nach Russell [Russell & Norvig, 2004, Kap. 2.4, S. 80]. Das interne Modell der Umwelt, sowie die interne Verwaltung des aktuellen Zustands, die Ziele und die Vorstellung der Auswirkung der Aktion in der Umwelt ist im Leistungselement verankert. Eine Änderung zum Beispiel der Nutzenfunktion wird im Lernelement in Abhängigkeit vom vorgegebenen Leistungsstandard berechnet.

Zustandes werden die Daten aus den Sensoren des Agenten verwendet. Das Leistungselement repräsentiert die Agentenfunktion, dabei wird das Leistungselement in Abhängigkeit vom zu lösenden Problem und der vorgegebenen Konfiguration (Aufbau der Sensoren und Aktuatoren) des Agenten definiert.
Das *Lernelement* erzielt die Änderungen im Wissensstand des Agenten, um sein Verhalten zu verbessern. Der lernende Agent soll die Möglichkeit haben, die Daten nicht nur vervollständigen und verwalten zu können, sondern sich auch die Auswirkungen auf seine Umgebung zu merken, um in ähnlichen Situationen sein Verhalten zu verbessern.
”Die *Kritik*-Komponente teilt dem Lernelement mit, wie gut sich der Agent im Hinblick auf den festgelegten Leistungsstandard verhält.” [Russell & Norvig, 2004, S. 80, Kap. 2.4.6]. Das zur Realisierung des Reinforcement-Learning benötigte *Belohnungsmodell* wird auf Grundlage der Kritik-Komponente und Leistungsstandards definiert. Der Agent beinhaltet einen intern eingebauten *Leistungsstandard*, der bewerten kann, wie gut bzw. schlecht die durchgeführte Interaktion für den Agenten in der Umgebung war. Wenn ein kleines Kind mit seiner Hand eine heiße Herdplatte berührt, wird die Antwort der Umgebung mithilfe von Rezeptoren, die sich in der Haut des Kindes befinden, ein Schmerzsignal gesendet. Aus dieser Interaktion wird das Kind lernen, dass eine heiße Herdplatte eine Gefahr für den Menschen darstellt.
Der *Problemgenerator* ist für den Explorationsschritt in der Umgebung verantwortlich. Der Agent kann zwar aus seiner Sicht rational handeln, seine Handlung im Allgemeinen kann aber nicht optimal sein. Der Explorationsschritt gibt dem Agenten eine neue Herausforderung und prüft nach neuen, möglichst besseren Lösungen.
In dieser Arbeit erfolgt das Lernen im Onlinebetrieb, was eine besondere Herausforderung darstellt. Eine ganze Kette von Aktionen, also ein Plan, der den Agenten zum vorgegebenen Ziel steuert, soll während

der Ausführung der Mission erlernt werden.
Der Lernprozess findet im Markowschen Entscheindungsproblem bzw. -prozess (MDP von engl.: Markov decision process) statt. In Abbildung 2.3 ist das Grundkonzept für die Interaktion zwischen Agent und Umgebung im MDP dargestellt. Zu jedem Zeitpunkt t befindet sich der Agent im Zustand $s_t \in S$ und hat eine Palette von möglichen Aktionen $A(s_t)$, die er auswählen kann. Der Agent bekommt für jede durchgeführte Aktion a_t eine Antwort von der Umgebung in Form eines elementaren "lokalen" Nutzens r_t (von engl.: reward) (gut, weniger gut, schlecht). Der Nutzen repräsentiert die Messung des Erfolgs. Der Agent hat die Möglichkeit, die gesammelten Erfahrungen (elementarer Nutzen) von Einzelschritten zu akkumulieren und die globale Erfahrung, wie das Ziel zu erreichen ist, in Form einer Nutzenfunktion zu erlernen. Mithilfe der Nutzenfunktion "wird die 'globale' Definition der Rationalität - in der die Agentenfunktionen als rational bezeichnet werden, die die höchste Leistung aufweisen - in eine 'lokale' Bedingung für den Entwurf rationaler Agenten umgewandelt, die in einem einfachen Programm ausgedrückt werden kann." [Russell & Norvig, 2004, S. 78, Kap. 2.4.5].

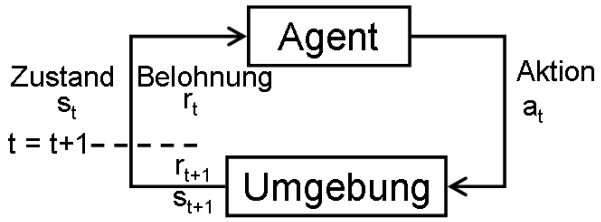


Abbildung 2.3: Interaktion zwischen dem Agenten und seiner Umgebung. Die wichtigsten Bestandteile sind dabei der aktuelle Zustand s_t, die ausgewählte Aktion a_t und der resultierende Nutzen r_t. Der Agent entscheidet sich situationsbasiert für eine Aktion a_t, die er ausführt. Dadurch geht der Agent in den neuen Zustand s_{t+1} über und beobachtet den von der Umgebung erzeugten Nutzen r_{t+1} für die ausgeführte Aktion. Der Vorgang wird abgeschlossen, und der Agent befindet sich im neuen Zeitschritt $t + 1$. Von da aus wird der gleiche Vorgang durchgeführt.

2.3 Architektur des MCP-Agenten

Die erste Art der rationalen Agenten wird zur Lösung des Mountain-Car-Problems entwickelt und zur Untersuchung der vorgeschlagenen Erweiterungen des Lernelements verwendet. Alle Versuche werden in der Simulation durchgeführt. Das Mountain-Car-Problem ist dem Buch von Sutton und Barto [Sutton & Barto, 1998, Kap. 8.4, S. 214] entnommen. Der Mountain-Car-Agent ist auf die Lösung des Mountain-Car-Problems ausgerichtet. Das Ziel des Agenten im Mountain-Car-Problem ist es aus einem Tal über den Berg auf die rechte Seite zu fahren. Das zu erreichende Ziel ist in Abbildung 4.1(b) durch einen roten Punkt gekennzeichnet. Dabei reicht die Kraft des Motors nicht aus, um direkt den Berg hinaufzufahren. Der Agent soll auf der linken Seite des Tals zusätzlichen Schwung holen, um das Tal verlassen zu können. Dies ist ein dynamisches Problem, eine Art Benchmark-Problem für die Reinforcement-Learning-Methoden.

2.3.1 Umgebung

Der MCP-Agent wird basierend auf dem Mountain-Car-Problem (MDP) als Auto definiert. Der interne Zustand des Agenten besteht aus der aktuellen Koordinate $x \in [-1.2, 0.6]$ und der aktuellen Geschwindigkeit des Autos $\dot{x} \in [-0.07, 0.07]$ und ist gleich dem globalen Zustand. Der Zustandsraum ist in MDP zweidimensional: $(x, \dot{x}) \in [-1.2, 0.6] \times [-0.07, 0.07]$. Der Startzustand liegt in der tiefsten Stelle des Tals mit der Geschwindigkeit Null, also an der Stelle $(x^{Start}, \dot{x}) := (-0.523, 0)$. Das Ziel ist es, mit beliebiger Geschwindigkeit $x^{Ziel} = 0.6$ zu erreichen.
Die Höhe im Tal $f(x)$ ist latent im Modell durch die Schwerkraft des Agenten versteckt. Diese Information wird in Form der Geschwindigkeit des Agenten mitgeteilt. Je nach Steuerungsbefehl werden die Zustände des Agenten neu berechnet. Dieser Schritt wird mit einem Differentialgleichungssystem in Abhängigkeit von der gewählten Steuerung (vorwärts beschleunigen, rückwärts beschleunigen, nicht beschleunigen) modelliert. Die Position des Autos wird mithilfe der kinematischen Gleichung berechnet, siehe Gleichung (2.1). Zu jedem Zeitpunkt wirkt die Schwerkraft auf den Agenten, die Steuerung wird mithilfe des Steuerungssignals a_t zu jedem Schritt t durchgeführt, siehe Gleichung (2.2). Diese Gleichungen stammen aus [Sutton & Barto, 1998, Kap. 8.4, S. 214].

$$x_{t+1} = x_t + \dot{x}_{t+1} \tag{2.1}$$

$$\dot{x}_{t+1} = \left\lfloor \dot{x}_t + \underbrace{0.001 \cdot a_t}_{\text{Steuerung}} - \underbrace{0.0025 \cdot \cos(3 \cdot x_t)}_{\text{Schwerkraft}} \right\rfloor \tag{2.2}$$

Der Agent bezieht seine Vorstellung von der Umgebung aus einer erlernten Nutzenfunktion.

2.3.2 Sensoren

Der Mountain-Car-Agent besitzt neben der Odometrie keine zusätzlichen Sensoren. Es wird angenommen, dass die aktuelle Position und Geschwindigkeit des Agenten zu jedem Zeitpunkt durch die eingeführten Formeln vorhersehbar ist.

2.3.3 Aktuatoren

Das Auto kann mit einem konstanten Wert vorwärts oder rückwärts beschleunigen und ohne Beschleunigung rollen. Zusätzlich wird das Auto von der Schwerkraft beschleunigt. Die Steigung der Fahrbahn ist abhängig von der Position des Autos und beeinflusst auch die aus der Schwerkraft resultierende Beschleunigung. Der Aktionsraum, also die Menge der möglichen Steuerungssignale, enthält in jedem Zustand drei mögliche Aktionen: $a_t \in A \in \{-1, 0, 1\}$. $a_t = -1$ bedeutet, dass das Auto rückwärts, also nach links, beschleunigt. $a_t = 0$ bedeutet Leerlauf, es wird weder rückwärts noch vorwärts beschleunigt. $a_t = 1$ bedeutet, dass der Vorwärtsgang eingelegt ist, und das Auto in Richtung des vorgegebenen Ziels beschleunigt, also nach rechts. Diese Aktionen werden für die Bestimmung des neuen Zustands des MCP-Agenten angewendet.

2.4 Navigationsproblem in der Robotik

”Mobile autonome Robotik ist ein junges und aktuelles Forschungsgebiet. Gleichzeitig ist es auch eines der faszinierendsten Themengebiete der heutigen Zeit. Wir alle träumen von einem vollständig autonomen Roboter, der selbständig alle seine Aufgaben erfüllt. Oder besser noch: der seine eigenen Aufgaben

formuliert um unser Leben schöner zu gestalten.” [Mellmann, 2011, Kap. 1, S. 1].
Der Roboter-Agent soll das in der vorliegenden Arbeit definierte Navigationsproblem unter vorliegenden Rahmenbedingungen lösen. Der Agent soll ausgehend von einem festgelegten Startpunkt einen vorgegebenen Zielpunkt erreichen. Das Modell der Umgebung ist nicht bekannt. Der Agent soll autonom handeln, also soll seine Fahrt kollisionsfrei sowie ungefährlich für ihn selbst und seine Umgebung sein. Der vom Agenten verfolgte Weg soll möglichst optimal sein, so dass das Ziel in möglichst kurzer Zeit erreicht wird.

2.4.1 Arten der Navigation

Die Navigation wird in eine lokale und eine globale Navigation unterteilt. Die Sensorik für die lokale Navigation muss die Informationen liefern, die mindestens auf die befahrbare Fläche schließen lassen, damit der Roboter sich sicher bewegen kann. Die globale Navigation benötigt ein globales Umgebungsmodell und planende Fähigkeiten [Stenzel, 2002].
Die *globale Navigation* benötigt globale Informationen wie die Position des Agenten bzw. den internen Zustand, die Position des Ziels sowie ein Modell (Karte) der Umgebung, um globale Pläne erstellen zu können. Diese Art von Navigation wird in der vorliegenden Arbeit durch das Leistungselement realisiert. Vom Leistungselement wird die benötigte Umgebungsrepräsentation verwaltet, und wenn nötig sequentiell aufgebaut. Der interne Zustand des Agenten in der internen Umgebungsrepräsentation wird aus den vorgegebenen Sensordaten extrahiert und zur Bestimmung neuer Aktionen verwendet.
Die *lokale Navigation* verarbeitet die Informationen aus der lokalen Umgebung (aus den lokalen Sensordaten), ohne die globalen Informationen (interner Zustand) zu berücksichtigen. Die lokale Navigation wird auch als Reflex-Agent bezeichnet, siehe [Russell & Norvig, 2004, Kap. 2.4.2, S. 72]. Diese Art der Navigation wird in der vorliegenden Arbeit als reaktive Fähigkeit realisiert und in den beiden Agentenarten (RL-Agent und DPA-RL-Agent), die auf die Lösung des Navigationsproblems ausgerichtet sind, angewendet. Diese reaktive Fähigkeit bildet die Rohdaten von den Sensoren direkt auf die Befehle für die Motorsteuerung ab.

2.4.2 Navigationsproblem

Das gesamte Navigationsproblem besteht aus vier Fragen, die sich der Agent stellt [Leonard & Durrant-Whyte, 1991]:

- Wo bin ich im Moment?
- Wo soll ich hin?
- Wie gelange ich dorthin?
- Wo bin ich gewesen?

Die erste Frage kann dann beantwortet werden, wenn der Agent über eine gewisse Vorstellung von seiner Umgebung verfügt. Die Lokalisation kann relativ oder absolut erfolgen. In der vorliegenden Arbeit werden Odometriedaten des Roboters zur Lokalisation angewendet. Die zweite Frage ist dann einfacher zu beantworten. Sie kann mit ”zum Ziel gelangen” beantwortet werden. Das Ziel muss nur im benötigten Format beschrieben werden.

Die dritte Frage ”Wie gelange ich dorthin?” kann erst dann mithilfe von z. B. Erstellen eines Plans[4] beantwortet werden, wenn der Agent wiederum eine gewisse Vorstellung von seiner Umgebung hat. Wenn kein Modell der Umgebung vorliegt, muss der benötigte Plan in mehreren Episoden erlernt werden. Dieser Weg wird in der vorliegenden Arbeit verfolgt.
Für die Beantwortung der vierten Frage ”Wo bin ich gewesen?” wird die benötigte Repräsentation der Umgebung während der Ausführung der Mission erfasst und gespeichert. Die in der vorliegenden Arbeit angewendeten Methoden benötigen keinen aufwändigen Schritt der Erstellung einer metrischen Karte. Während der Ausführung der Mission wird das Modell der Umgebung parallel vom Agenten erstellt und angewendet, so dass gleichzeitig das Ziel verfolgt wird, und die Exploration durchgeführt wird.

2.4.3 Analogie zu Problemen moderner KI

Die Zukunft für rationale Agenten liegt nach Meinung der Autoren aus [Russell & Norvig, 2004, S. 1176, Kap. 27.1] in der weiteren Erforschung der folgenden Teilgebiete der KI.

1. ”Zusammenarbeit mit der Umgebung durch Sensoren und Aktuatoren”
2. ”Beobachtung des Zustandes der Umwelt”
3. ”Projektion, Auswertung und Auswahl zukünftiger Aktionsfolgen”
4. ”Nutzen als Ausdruck von Prioritäten”
5. ”Lernen”

Die genannten Teilgebiete sind auch im Navigationsproblem vertreten. Es wird gezeigt, dass die vorliegende Arbeit einen Beitrag zur Lösung vieler allgemeiner Probleme moderner künstlicher Intelligenz in allen oben genannten Teilgebieten der KI leistet, indem der lernende rationale Agent zur Lösung des Navigationsproblems entwickelt wird.
”Zusammenarbeit mit der Umgebung durch Sensoren und Aktuatoren”: In der Geschichte der künstlichen Intelligenz waren bis auf einzelne Ausnahmen die KI-Systeme so aufgebaut, dass Menschen die Eingaben und die Ausgaben des Agenten interpretieren mussten und das Schließen und Planen auf höchster Abstraktionsebene größtenteils fehlte [Russell & Norvig, 2004, Kap. 27.1, S. 1176]. Die in der vorliegenden Arbeit entwickelten Agenten besitzen die Fähigkeit die Daten über das vorhandene Wahrnehmungsmodell selbst zu sammeln und selbst zu interpretieren. Der Plan wird durch Ausprobieren der möglichen Wege vom Agenten selbst erzeugt. Der Mensch soll den Roboter-Agenten nur vom Ziel zum Startpunkt zur Durchführung der nächsten Episode tragen. Die entwickelte Lösung hat aber ein großes Potenzial, so dass auch dieser Schritt autonom durch den Agenten ausgeführt werden kann.
”Beobachtung des Zustandes der Umwelt”: Die Beobachtung kann auf unterschiedliche Arten des gesammelten bzw. vorhandenen Wissens (z. B. erstelltes oder vorhandenes Umgebungsmodell, Sensordaten) basieren. Umgebungsrepräsentation kann in unterschiedlicher Art und Weise im Navigationsproblem erfolgen. Die Effizienz des Lernens kann erhöht werden, wenn Informationen nur an den wichtigen Entscheidungsknoten gespeichert und verarbeitet werden.
”Projektion, Auswertung und Auswahl zukünftiger Aktionsfolgen”: Die einzelnen Aktionsverläufe bestehen teilweise aus Hunderten von Einzelschritten. Die Menschen kommen nur durch die Einführung

[4] einer Kette aus Zuständen und Aktionen, die den Agenten vom Start zum Ziel führt

einer hierarchischen Struktur des Verhaltens zurecht. Das hierarchische verstärkende Lernen ist nach Meinung der Autoren Russell und Norvig ein wichtiger Begriff für die Zukunft der modernen KI. Das hierarchische Reinforcement Lernen wird im DPA-RL-Agenten als Grundkonzept für das Erlernen der komplexen Strategie auf Grundlage der abstrakten Aktionen angewendet.
”Nutzen als Ausdruck von Prioritäten”: Das Treffen von rationalen Entscheidungen, die auf der Maximierung des Nutzens basieren, ist ein allgemeines Konzept. Mit diesem Konzept können viele Probleme vermieden werden, die in einem rein zielbasierten Ansatz auftreten.[5] Das Belohnungsmodell, auf dessen Grundlage die Nutzenfunktion aufgebaut ist, kann einfach sein. Hingegen kann die eigentliche Nutzenfunktion sehr komplex sein [Russell & Norvig, 2004, Kap. 2.4.5, S. 78]. Die Nutzenfunktion ist einer der grundlegenden Begriffe der vorliegenden Arbeit. Die existierenden Methoden aus dem Bereich des Reinforcement-Learning werden zur Konstruktion der Nutzenfunktion angewendet und erweitert.
”Lernen”: Keiner der in der vorliegenden Arbeit entwickelten Agenten besitzt Vorwissen über seine Umgebung, so dass sie gezwungen werden immer wieder basierend auf den aktuellen Informationen ihre Nutzenfunktion zu aktualisieren und somit auch zu lernen. Die Entwicklung des rationalen Agenten mit der Fähigkeit zum ”Lernen” ist eins der verfolgten Ziele der vorliegenden Arbeit und ein sehr wichtiges Gebiet moderner künstlicher Intelligenz.

2.5 Architektur des Roboter-Agenten

”Ein Roboter-Agent könnte Kameras und Infrarotbereichsmelder als Sensoren verwenden und verschiedene Motoren als Aktuatoren” [Russell & Norvig, 2004, S. 55, Kap. 2.1]. Der Roboter-Agent in der vorliegenden Arbeit beinhaltet die vorgegebene Konfiguration der Sensoren und Aktuatoren des echten Scorpion-Roboters und ist auf die Lösung des Navigationsproblems ausgerichtet. In der vorliegenden Arbeit werden zwei Agentenarten zur Lösung von Navigationsproblemen entwickelt: der RL-Agent und der DPA-RL-Agent. Die beiden Agentenarten sind eine Spezialisierung des Roboter-Agenten. Die Architektur des Roboter-Agenten und die Umgebung, in der der Agent agiert, bleiben für den RL-Agenten sowie den DPA-RL-Agenten gleich. Das Agentenprogramm, das auszuführende Aktionen bestimmt, und die Wissensrepräsentation unterscheiden sich je nach Agententyp.

2.5.1 Umgebung

Die Umgebung für die globale Navigation ist in dieser Arbeit ein Bewegungsareal des Roboter-Agenten. Der RL-Agent muss die Position des Ziels im Vorfeld nicht kennen, erhält aber ein Signal, wenn das Ziel erreicht ist. Der DPA-RL-Agent beinhaltet schon am Anfang seiner Mission die Position des Ziels in seinem Koordinatensystem.
Der interne Zustand im Zustandsraum wird im Navigationsproblem durch die Odometrie, einen Mechanismus zur Berechnung der aktuellen Position im Koordinatensystem des Startpunktes des Roboters, bestimmt. Der Zustand besteht aus der zweidimensionalen Position des Agenten in der Ebene und der Orientierung des Agenten (x, y, o). Der Ursprung des Koordinatensystems wird meistens auf die Startposition des Agenten gelegt, das Ziel wird in diesem Koordinatensystem definiert.

[5] Solche Probleme können durch konkurrierende Ziele entstehen, auf deren Grundlage es schwer ist eine richtige Strategie für den Agenten zu erstellen.

2.5.2 Sensoren

Ein autonomer Roboter-Agent muss die Fähigkeit besitzen in nicht genau beschreibbaren Umgebungen agieren zu können. Zum Beispiel sind die Motoren des Roboters nicht exakt, die Umgebung ist unbekannt, was in einer Unvorhersehbarkeit resultiert. Die Unvorhersehbarkeit wiederum impliziert die Notwendigkeit von Sensoren zur Wahrnehmung der Umgebung, die lokale und globale Informationen über die Umgebung liefern. Die Sensoren bringen weitere Schwierigkeiten mit sich, sind ungenau und liefern Fehler. Außerdem sind Methoden zur Extraktion der nützlichen Informationen aus den Sensordaten erforderlich [Arras & Siegwart, 2000].
In der vorliegenden Arbeit werden reale Scorpion-Roboter der Firma Evolution Robotics sowie das zur Simulation entwickelte Modell des Scorpion-Roboters verwendet. Der Scorpion-Roboter beinhaltet den Odometrie-Sensor, eine Weitwinkelkamera, Infrarotsensoren sowie einen Kontaktsensor zur Erfassung seiner Umgebung, siehe Abbildung 2.6.
Der Roboter-Agent verfügt über eine bestimmte Kombination an Sensoren des echten Scorpion-Roboters. Die verwendeten Sensoren sind ausgewählte Abstandssensoren und die Odometrie. Die vom Roboter-Agenten verwendeten Sensoren werden zur Bestimmung des internen Zustands und Aufbaus des internen Modells der Umgebung des Roboter-Agenten angewendet. Die Aktuatoren sind die Motoren an zwei Rädern des Scorpion-Roboters.

2.5.2.1 Odometrie

Der Scorpion-Roboter beinhaltet die Odometrie, die vom Roboter-Agenten in der Realität angewendet bzw. in der Simulation simuliert wird. Die Odometrie gibt die aktuelle Position und Orientierung des Agenten im Koordinatensystem mit dem Ursprung in der Startposition des Agenten an. Bei den Odometriedaten wird angenommen, dass die gelieferten Daten wenige Fehler aufweisen.
Ein gutes Navigationssystem soll den aktuellen Zustand sicher bestimmen können, also sich selbst lokalisieren können. Der Selbstlokalisierungsschritt, die sichere Feststellung des aktuellen Zustands des Agenten im Umgebungsmodell, wird zur Bestimmung der besten Aktion in der aktuellen Situation im Leistungselement benötigt. Im Rahmen der vorliegenden Arbeit werden auch Studien im Bereich der Selbstlokalisierung durch die Anwendung des Partikel-Filter-Ansatzes, siehe auch [Thrun et al., 2000], für die Verbesserung der Datenqualität aus der Odometrie durchgeführt, siehe zum Beispiel die Diplomarbeiten [Eickhoff, 2009] und [Zhigun, 2011]. Als Vorbereitung für die Lokalisierung mithilfe von visuellen Landmarken wurden Untersuchungen zur Objektklassifikation mithilfe von Support-Vektor-Maschinen durchgeführt, siehe zum Beispiel [Sluzhivoy et al., 2008] oder die Diplomarbeiten [Sluzhivoy, 2008], [Lisowski, 2009]. Dieser Zweig der Arbeit ist aber nicht weiter verfolgt worden, da die benötigten Daten, die von der Odometrie des echten Scorpion-Roboters gewonnen werden, relativ genau für kleinere, in der vorliegenden Arbeit verwendete Szenarien sind.
Die Odometriedaten können je nach Sichtweise als Daten von einem eigenständigen Sensor, oder als interner Informationsfluss betrachtet werden. In der vorliegenden Arbeit wird die Odometrie als eigenständiger Sensor eingesehen. In allen Abbildungen des Agenten beinhaltet das Kästchen ”Sensoren” auch die Daten aus der Odometrie des Agenten, siehe Pfeil vom Kästchen Sensoren zum Leistungselement in Abbildung 2.2.

2.5.2.2 Infrarotsensoren

Die Infrarotsensoren (IR-Sensoren) messen die Abstände zu Gegenständen mit einer vorgegebenen maximalen Sichtweite des Sensors. Mit IR-Sensoren wird ein lokaler Abdruck der Umgebung gebildet und

so die lokale Navigation realisiert.
Der Scorpion-Roboter verfügt insgesamt über 20 IR-Sensoren zur Abstandsmessung, die an seiner äußeren Peripherie verteilt angebracht sind und einen breiten Rundumblick ermöglichen [EvolutionRobotics, 2004]. Die Gesamtzahl dieser Sensoren teilt sich in 13 analoge IR-Sensoren GP2D12 und 7 digitale IR-Sensoren GP2D15 auf. Analoge IR-Sensoren GP2D12 sind parallel zum Boden ausgerichtet und ermöglichen die Messung der Entfernung zu Objekten in der Ebene. Die 7 digitalen IR-Sensoren GP2D15 sind die nicht parallel zum Boden ausgerichteten Sensoren und sollen beispielsweise Gefahr, die von Treppen ausgeht, erkennen können.
Das Prinzip der Abstandserkennung der GP2D12-IR-Sensoren basiert auf der Messung der von Objekten reflektierten infraroten Strahlung. Hierfür ist der Sensor mit einem IR-Sender und -Empfänger ausgestattet. Der Sensor emittiert einen Impuls aus IR-Strahlung. Der Empfänger bestimmt anschließend die Energie der zurückgeworfenen Strahlung gleicher Frequenz. Die tatsächlich bestimmte Entfernung hängt also auch von Größe, Form, Orientierung und durchschnittlichem Reflexionsverhalten eines Objekts ab [EvolutionRobotics, 2004].
Die GP2D12-IR-Sensoren sind laut Spezifikation auf Entfernungsmessungen im Bereich von $10\ cm$ bis $80\ cm$ beschränkt. Messungen unterhalb dieses Intervalls resultieren in nicht korrekten Ergebnissen. Die mehr als $80\ cm$ entfernten Objekte können nicht mehr durch diese Sensorart erfasst werden [EvolutionRobotics, 2004].
Im Roboter-Agenten wurden sieben zum Boden parallel ausgerichtete Sensoren verwendet, siehe Abbildung 2.4. Die Wahl von nur sieben Sensoren in der Konfiguration des Roboter-Agenten ist durch allgemeine Anwendbarkeit bedingt. Zum Vergleich der Ergebnisse soll die Konfiguration der angewendeten Agenten gleich sein. Für viele Methoden können zahlreiche Sensoren ohne Einschränkungen und ohne großen Rechenaufwand angewendet werden. Es werden aber auch Methoden im Roboter-Agenten angewendet, bei denen eine exponentielle Abhängigkeit zwischen dem Rechen- und Kapazitätsaufwand und der Anzahl der Sensoren besteht.

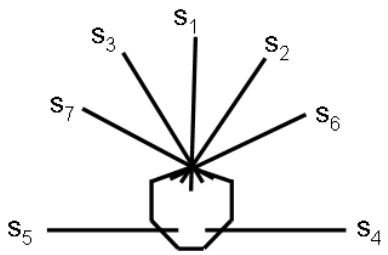


Abbildung 2.4: Roboter-Agent mit sieben verwendeten Sensoren

2.5.2.3 Virtueller Sensor: Kompass-Sinn

Der Roboter-Agent wird um einen virtuellen Sensor erweitert, der in der Konfiguration des Scorpion-Roboters nicht vorhanden ist. Dieser Sensor gibt zu jedem Zeitpunkt aus, in welcher Richtung sich das vorgegebene Ziel befindet. Durch einen so definierten Sensor wird die globale Information zugleich mit Odometriedaten in das System eingegeben. Dieser virtuelle Sensor kann nur dann angewendet werden, wenn die Position des Ziels vorhanden ist.
Der virtuelle Sensor Kompass-Sinn kann auf Grundlage der am Scorpion-Roboter angebrachten Kamera-Sensoren definiert werden und so von den Odometriedaten entkoppelt werden oder auch die Odometriedaten verarbeiten. Diese Flexibilität hat dazu beigetragen diesen virtuellen Sensor als eigenständige

Einheit zu betrachten.
Der Messwert des Kompass-Sinns wird als ein zum Ziel ausgerichteter, normierter Vektor repräsentiert, siehe rote Pfeile in Abbildung 2.5, und als Winkel in das System propagiert. Die tatsächliche Distanz zum Ziel wird durch diesen Sensor nicht erfasst.

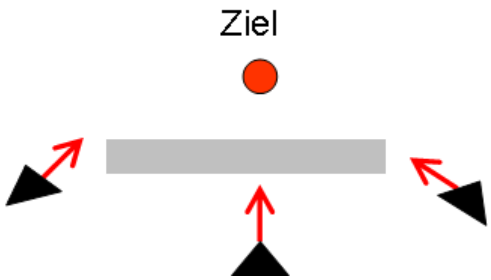


Abbildung 2.5: Richtungsangabe des Kompass-Sinns im Roboter-Agenten. Der Agent mit seiner Ausrichtung im Raum ist als ein Dreieck dargestellt. Der Kompass-Sinn ist der in Richtung Ziel ausgerichtete, normierte Vektor, der als roter Pfeil dargestellt ist.

2.5.3 Aktuatoren

Die Aktuatoren des Roboters sind sein Antriebssystem. Es besteht aus zwei nicht aneinander gekoppelten, fest montierten Antriebsrädern mit jeweils eigenem Motor. Zusätzlich ist ein in der senkrechten Achse und in der Radmittelachse frei drehbares Rad ohne Motor vorhanden, das zur Stabilisierung im Heckbereich dient. Der Scorpion-Roboter kann sich drehen, indem die motorisierten Räder mit unterschiedlichen Drehrichtungen und Geschwindigkeiten angetrieben werden. Der Scorpion-Roboter erwartet als Steuerbefehl zwei Geschwindigkeiten für die Motoren. Somit besteht eine Aktion des Roboter-Agenten aus zwei Komponenten. Diese Komponenten sind (ω, v), wobei mit ω die Drehgeschwindigkeit in Grad pro Minute und mit v die Vorwärtsgeschwindigkeit in cm pro Minute bezeichnet wird. In der Simulation wird das System im vorgegebenen Takt aktualisiert und die neue Position und Orientierung des Agenten durch Anwendung der Formeln aus der Kinematik berechnet.
Der Agent kann Aktionen unterschiedlicher Abstraktionsgrade ausführen. Unter Aktionen niedriger Abstraktionsgrade werden in dieser Arbeit die Aktionen verstanden, die elementare, diskrete Befehle an die Motorsteuerung senden, diese Befehle bleiben konstant und sind unabhängig von der lokalen Umgebung.
Die Aktionen niedriger Abstraktionsgrade werden als *diskrete Aktionen* bezeichnet. Die Aktionen höherer Abstraktionsgrade beinhalten ganze *reaktive Fähigkeiten* und werden als *abstrakte Aktionen* bezeichnet. Diese reaktiven Fähigkeiten können durch eine oder eine Verknüpfung von mehreren Kontrollfunktionen realisiert werden, sind autonom, reaktiv, können eine unterschiedliche Dauer haben, liefern Befehle auf die Motorsteuerung in Abhängigkeit von der aktuellen, lokalen Umgebung. Durch eine solche reaktive Fähigkeit wird die Vermeidung von Kollisionen realisiert und so auch die Autonomie des Agenten. Mithilfe dieser reaktiven Fähigkeiten wird die lokale Navigation realisiert.
Der Unterschied zwischen dem RL-Agenten und dem DPA-RL-Agenten liegt in der Anwendung verschiedener Aktionsräume. Der *RL-Agent* ist der Roboter-Agent mit einem Aktionsraum von niedrigem Abstraktionsgrad[6]. Dagegen ist der *DPA-RL-Agent* ein Roboter-Agent, der Aktionen von höherem Ab-

[6] zum Beispiel ”fahre 10 cm nach vorne”

Abbildung 2.6: Der in der Realität verwendete Roboter-Agent der Firma Evolution Robotics. Er besitzt eine Weitwinkelkamera, einen Kontaktsensor und IR-Sensoren, die parallel und orthogonal zum Boden ausgerichtet sind. Der Roboter besitzt Zähler auf den Motoren, die die Koordinate des Roboters in Bezug auf seine Startposition ausgeben (Odometriesensor). Es werden nur die 7 ausgewählten, parallel zum Boden ausgerichteten Sensoren und der Odometriesensor verwendet.

straktionsgrad[7] verwaltet. Der RL-Agent verwendet eine Aktionsmenge, die aus drei diskreten Aktionen (”Drehe rechts”, ”Drehe links”, ”Fahr geradeaus”) besteht und wird durch einen zweidimensionalen Vektor

$$\{(15°, 0), (-15°, 0), (0°, 10)\}$$

realisiert[8], so dass die direkten Steuerbefehle, die der Roboter-Agent für die Motorsteuerung benötigt, geliefert werden.
Der DPA-RL-Agent beinhaltet die Aktionsmenge, die aus abstrakten Aktionen besteht und durch reaktive Fähigkeiten realisiert wird. ”Zum Ziel fahren” ist die erste abstrakte Aktion, sie verarbeitet globale Informationen über das Ziel des Agenten in der Umgebung und realisiert die globale Navigation des Agenten. Die nächsten abstrakten Aktionen sind ”Hindernis links umfahren” und ”Hindernis rechts umfahren”. Diese abstrakten Aktionen können mithilfe unterschiedlicher Methoden realisiert werden.

2.6 Realitätsnahe Simulation

Im Rahmen der vorliegenden Arbeit ist eine realitätsnahe Simulation entstanden. Dieser Modellierungsschritt wird entwickelt, um die benötigten reaktiven Fähigkeiten des Roboter-Agenten zu bilden. Die in der realitätsnahen Simulation erzeugten, reaktiven Fähigkeiten werden in beiden Arten des Roboter-Agenten angewendet. Dieses Konzept ist in der Diplomarbeit von Thomas Köpsel [Köpsel, 2007] und im Artikel [Köpsel et al., 2007] bereits eingeführt. Die nahe Verknüpfung der Realität und Simulationsumgebung trägt entscheidend dazu bei, dass die in der Simulation erzeugten reaktiven Fähigkeiten des Roboter-Agenten ohne Schwierigkeiten in der realen Umgebung auf dem realen Scorpion-Roboter ohne

[7] zum Beispiel ”fahre zum Ziel”
[8] Die gewählten, konstanten Werte haben keine besondere Bedeutung. Diese Werte sollten nicht zu groß sein, damit der Agent gut manövrieren kann.

Nachbearbeitungsschritt angewendet werden können. Die für manche entwickelte Methoden benötigte Kalibrierung in der realen Umgebung kann in der realitätsnahen Simulation schneller und unkomplizierter durchgeführt werden. Durch das Konzept der realitätsnahen Simulation wird ein höherer Grad der Generalisierbarkeit erreicht.

2.6.1 Aufbau der realitätsnahen Simulation

In Abbildung 2.7 ist die Bildung der realitätsnahen Simulation des Roboter-Agenten skizziert. Durch die Farben werden die schon aus dem lernbasierten Agenten bekannten Elemente (Leistungselement, Sensoren, Aktuatoren) verdeutlicht, vergleiche Abbildung 2.2. Diese Elemente werden sowohl in der Realität als auch in der Simulation realisiert. Mit der Farbe rosa wird das Leistungselement (Roboter) gekennzeichnet, mit der Farbe hellgrün wird das Element Sensoren aus dem lernbasierten Agenten in der Abbildung der realitätsnahen Simulation markiert. Die Farbe gelb deutet in beiden Abbildungen auf das Element Aktuatoren hin. Für den Aufbau der realitätsnahen Simulation werden in der Realität erzielte

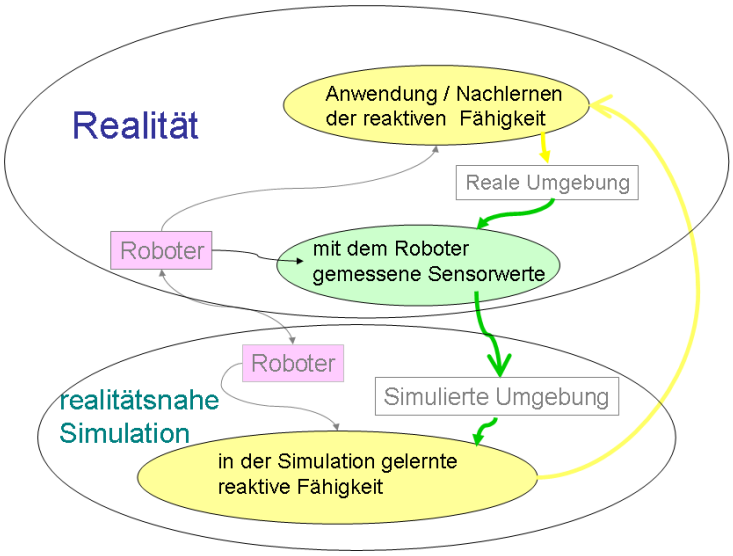


Abbildung 2.7: Ablauf der realitätsnahen Modellierung in der Simulationsumgebung für den Roboter-Agenten. Die Farben verdeutlichen die gleichen Elemente wie beim lernbasierten Agenten, siehe Abbildung 2.2

Sensorwerte angewendet, vergleiche hellgrüner Pfeil aus dem Feld ”Mit dem Roboter gemessene Sensorwerte”. Die Anwendung der in der Realität gemessenen Werte in der Simulation wird als realitätsnahe Simulation bezeichnet. Die so generierten reaktiven Fähigkeiten werden in den realen Scorpion-Roboter eingebunden. Somit entsteht ein Zyklus, der eine flexiblere Anpassung des Roboter-Agenten in der Simulation und in der Realität erlaubt.
Ein Beispiel der Anwendung des Roboter-Agenten in der Simulation und in der Realität ist in Abbildung 2.8 und 2.9 dargestellt. Die in der Simulation erzielte Fähigkeit im Parcours zu fahren wird in der Realität angewendet.

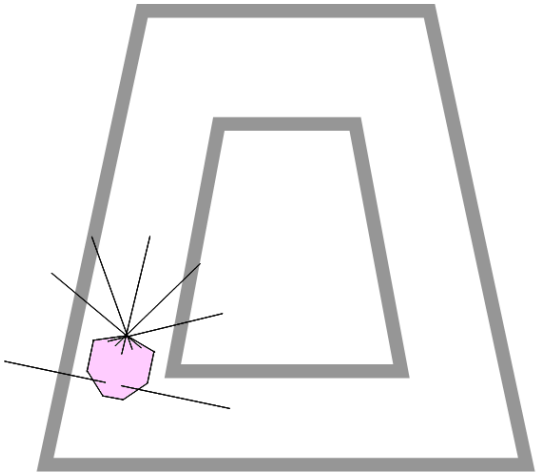


Abbildung 2.8: Agent in einem Beispielszenario in der Simulation

Abbildung 2.9: Agent in einem Beispielszenario in der Realität

2.6.2 Realisierung des Konzepts

Die simulierten IR-Sensoren des simulierten Roboter-Agenten sind wie beim realen Scorpion-Roboter zugeordnet. Sie geben die in der realen Umgebung gemessenen Sensorwerte aus. Diese Sensorwerte werden vorher in der realen Umgebung gemessen und in einer Tabelle gespeichert, siehe ”Mit dem Roboter gemessene Sensorwerte” in Abbildung 2.7. Für die Bestimmung des Modells ”Mit dem Roboter gemessene Sensorwerte” wird ein Diskretisierungsschritt benötigt, da nur eine endliche Anzahl an Messungen möglich ist. Jeder der IR-Sensoren liefert in der Realität ein Signal, dessen Werte in einem Intervall liegen. Das Modell ”Mit dem Roboter gemessene Sensorwerte” wird für diskrete Werte (Punkte) aus dem Intervall gebildet. Die vorhandenen diskreten Werte sollen dann interpoliert werden. Die interpolierten Werte approximieren die tatsächlich benötigten realen Sensorwerte.
Das Intervall für die Sensorwerte wird durch punktuelle Messungen in der Realität mit einem diskreten Abstand von $5\ cm$ approximiert. Die Sensorwerte in der Realität werden für folgende Distanzen (Punkte im Intervall) ermittelt.

$$d^{ideal} \in \{10, 15, 20, 25, \ldots, 80\} \tag{2.3}$$

Diese Distanzen sind die exakten Abstände vom Scorpion-Roboter zum Hindernis, die mithilfe eines Lineals gemessen werden. Insgesamt werden für jeden Sensor und jeden Punkt des Intervalls 1000 Messwerte ermittelt. So wird eine Tabelle mit einer Verteilung der Sensorwerte bestimmt. Die Werte, die zwischen den ermittelten Punkten liegen, werden für die Bildung der simulierten Sensorwerte linear interpoliert.
Mithilfe des dargestellten Beispiels wird der Schritt der linearen Interpolation erläutert. Ein Ausschnitt der Verteilung der in der Realität gemessenen Werte für den Sensor s_1, siehe Abbildung 2.4, und ein Ausschnitt der idealen Distanz bis zum Hindernis $d^{ideal} = 15\ cm$ sieht wie folgt aus:

$$f_{s_1}(d^{ideal} = 15) := \{19.18, 14.61, 15.0, 18.87, 14.69, \ldots\}$$

Wenn die ideal gemessene Distanz des simulierten Roboters den Wert $d^{ideal} = 17\ cm$ annimmt, muss der Interpolationsschritt durchgeführt werden, da keine Werte für die realen Messungen für diese Distanz in der vorliegenden Tabelle vorliegen. In der vorliegenden Tabelle sind Werte für die ideale Distanz von $15\ cm$ und $20\ cm$ vorhanden, vergleiche (2.3), $17 \in [15, 20]$. Sei $d_{min} = 15\ cm$ und $d^{ideal}_{max} = 20\ cm$. Für

die in der Tabelle vorhandenen, diskreten Werte $f_{s_1}(d_{min}^{ideal} = 15\ cm)$ und $f_{s_1}(d_{max}^{ideal} = 20\ cm)$ werden zwei tatsächlich gemessene Werte per Zufall ausgewählt. Der approximierte Wert der Distanz, die der Sensor s_1 in der Realität für die in der Simulation gemessene ideale Distanz von $17\ cm$ liefern könnte, wird durch echte reale Werte wie folgt linear approximiert:

$$f_{s1}(d^{ideal} = 17) = \frac{(f_{s1}(d_{max}^{ideal}) - f_{s1}(d_{min}^{ideal})) \cdot (d^{ideal} - d_{min}^{ideal})}{\underbrace{5}_{\text{diskreter Abstand zwischen den gemachten Messungen}}} + f_{s1}(d_{min}^{ideal})$$

Sei $f_{s1}(20) = 22.11, f_{s1}(15) = 19.18$, dann wird der reale Wert, den der Sensor s_1 ausgeben könnte, für die ideale Distanz $d^{ideal} = 17\ cm$ wie folgt approximiert: $f_{s1}(d^{ideal} = 17) = \frac{(22.11-19.18)\cdot(17-15)}{5} + 19.18 = 20.352$. Der simulierte Roboter wird für den ideal gemessenen Abstand zum Hindernis d^{ideal} den Wert $f_{s1}(d^{ideal} = 17\ cm)$ erhalten. Dieser Wert nähert sich dem tatsächlichen Wert aus dem Sensor s_1 für die ideale Distanz $17\ cm$.

2.7 Ausgangslage

Es existieren zahlreiche Architekturen für Agenten, die in bekannten Umgebungen handeln. Bekannte Vertreter solcher robotischen Kontrollarchitekturen sind z. B Atlantis, AuRA, 3T und Saphira, siehe [Arkin, 1998] und [Murphy, 2000]. Das in der vorliegenden Arbeit gestellte Problem geht aber von einer unbekannten Umgebung aus und fordert vom Agenten die Fähigkeit zum Lernen.
An dieser Stelle wird ein kurzer Überblick über bekannte Architekturen für Agenten gegeben. Dieses Unterkapitel hat keinen Anspruch auf Vollständigkeit, da die Anzahl der Architekturen mit jedem Jahr immer weiter wächst. Es wird aber eine kleine Übersicht gegeben, auf welchen Architekturarten die entwickelten Agententypen basieren, und welche konkreten Realisierungen der genannten Architekturarten existieren. Zuerst werden die hybriden Architekturen spezifiziert, die reaktive und deliberative Schichten beinhalten. Die BDI-Architektur[9] ist auch eine hybride Architektur.
Ein vollständiger Agent muss in der Lage sein unter Zeitmangel handeln zu können sowie zeitaufwendige Planungsschritte durchführen zu können. Dies wird von einer hybriden Architektur[10] nach [Russell & Norvig, 2004] ermöglicht.
Die hybride Software-Architektur, in der Robotik für zielbasierte Agenten auch hybride Architektur genannt, ist eine der am meisten verbreiteten Architekturarten in autonomen Systemen. Ein typisches Hybrid-Modell besteht aus drei Schichten. Diese Schichten kommunizieren nur mit der jeweils benachbarten Schicht. Die erste Schicht bildet dabei eine Planungskomponente. Die Planungskomponente beinhaltet ein internes Umgebungsmodell und verwaltet den internen Zustand des Agenten. Die Planungskomponente erstellt einen Plan, der aus einer Aktionskette besteht. Diese Aktionskette steuert den Agenten zum globalen Ziel. Die zweite Schicht verbindet die erste und dritte Schicht miteinander. In der Literatur wird diese Schicht als Sequencer bezeichnet. Der Sequencer zerlegt die von der Planungskomponente bestimmte Aktionskette noch einmal und aktiviert diese Aktionen in der dritten Schicht. Die dritte Schicht führt die von der Planungskomponente bzw. dem Sequencer ermittelte Aktion aus und wird als reaktive Schicht bezeichnet [Wikipedia, 2011b]. Es existieren zahlreiche Konzepte für solche hybride Architekturen, zum Beispiel 4D/RCS in [Albus, 2002] und Autonomous Animat in [Donnart & Meyer, 1995].

[9] BDI steht für Beliefs (Überzeugungen), Desires (Wünsche), Intention (Absicht) [Georgeff et al., 1999].
[10] Die Kombination reaktiver und planender Techniken wird häufig als hybride Architektur bezeichnet.

Die wissenschaftliche Grundlage der BDI-Architekturen ist die Theorie des ”praktischen Schlussfolgerns”. Die BDI-Architektur beinhaltet eine interne Weltrepräsentation (Beliefs), Vorstellungen über die Ziele (Desires) und den Plan selbst (Intentions) und hat damit eine konkretere Struktur. Diese Architektur wurde von Michael Bratman im Rational Agency Project des Stanford Research Institute als philosophisch motivierte Theorie menschlicher Handlung und Entscheidungsfindung entwickelt und bildet die hybride Architektur [Bratman, 1987], [Bratman et al., 1988].
Die in [Burkhard et al., 2002] präsentierte Doppelpass-Architektur (DPA) gehört zur Klasse der hybriden Architekturen und beruht auf dem Konzept der BDI-Architektur. Diese Architektur ist aus langjährigen Erfahrungen mit den klassischen Agentensystemen entstanden. Die DPA bildet das Grundgerüst für den entwickelten DPA-RL-Agenten, der die Fähigkeit beinhaltet, das fehlende Wissen über die Umgebung zu vervollständigen und anzuwenden.

Kapitel 3

Reinforcement-Learning-Methode

In der vorliegenden Arbeit wird der Agent mit einem ”sequentiellen Entscheidungsproblem” konfrontiert, ”wo der [Gesamt]Nutzen eines Agenten von einer Folge von Entscheidungen abhängig ist” [Russell & Norvig, 2004, Kap. 17, S. 749]. Dieses Entscheidungsproblem wird als Markow-Entscheidungsproblem (engl.: Markow Decision Process, MDP) formuliert. Das Ziel des Agenten liegt im Erlernen des schnellsten Wegs zum Ziel. Dieser Weg besteht aus einer Kette von angefahrenen Zuständen und ausgeführten Aktionen im MDP. Die Nutzenfunktion wird auf Grundlage der Gesamtbelohnung für die einzelnen Zustände der vorgegebenen Kette von Zuständen und Aktionen gebildet. Aus der optimalen Nutzenfunktion[11], die die Gesamtbelohnung der vom Agenten besuchten Zustände repräsentiert, kann die gesuchte Entscheidungskette gebildet werden. Die optimale Nutzenfunktion soll aus den gemachten Erfahrungen des Agenten erlernt werden.
Kapitel 3.1 verschafft eine kleine Übersicht über die existierenden Lernkonzepte in der künstlichen Intelligenz. Dabei wird begründet, warum der Reinforcement-Learning-Ansatz in der vorliegenden Arbeit angewendet wird.
In Kapitel 3.2 wird zuerst das allgemeine Prinzip des Reinforcement-Learnings eingeführt. Dafür werden wichtige Begriffe wie MDP und Nutzenfunktion und die Erläuterung der Bedeutung der Optimalität für die Nutzenfunktion eingeführt.
Der Reinforcement-Learning-Ansatz besteht aus einer ganzen Familie von Algorithmen. Der SARSA-Algorithmus ist eine mögliche Realisierung des allgemeinen Konzepts. Die Eigenschaft, die den SARSA-Algorithmus auszeichnet, ist die Möglichkeit im Onlinebetrieb zu lernen. Die Herleitungen der Update-Regel sowie des SARSA-Algorithmus für das endliche MDP (mit diskretem Zustands- und Aktionsraum) werden in Kapitel 3.3 eingeführt. Die Bedeutung der einzelnen Lernparameter wird in diesem Kapitel erläutert.
Die in der vorliegenden Arbeit behandelten Probleme erfüllen die Bedingung, dass der Zustandsraum eine endliche Anzahl an Zuständen beinhaltet, nicht. Die Definition der Nutzenfunktion für das endliche MDP und der zur Bestimmung der gesuchten Nutzenfunktion eingeführte SARSA-Algorithmus sollen für den kontinuierlichen Zustandsraum erweitert werden. Dieser Schritt wird in Kapitel 3.4 behandelt. Die unterschiedlichen Methoden für die Diskretisierung des Zustandsraumes werden in diesem Kapitel präsentiert. Die Approximation der Nutzenfunktion wird durch eine lineare Funktion bei Anwendung der geeigneten Diskretisierung des Zustandsraumes realisiert. Mit dem SARSA-Algorithmus im kontinuierlichen Zustandsraum wird das Kapitel abgeschlossen.
Das Semi-Markow-Entscheidungsproblem (SMDP) mit modifizierter Lernregel wird in Kapitel 3.5 eingeführt. Das SMDP kommt im DPA-RL-Agent zur Geltung.
Die in Kapitel 3.6 eingeführten Messkriterien werden zur Analyse der mit unterschiedlichen Konfigurationen bestimmten Ergebnisse verwendet. Die Aussagekraft der Vergleiche der erzielten Ergebnisse

[11] In anderen Literaturquellen wird die Nutzenfunktion für die Familie der Reinforcement-Learning-Methoden als Bewertungsfunktion bezeichnet.

beruht auf diesen Messkriterien.
Für unterschiedliche Problemarten werden in Kapitel 3.7 unterschiedliche Codierungsarten des Zustandsraumes untersucht, um die erzielten, optimalen Konfigurationen in den folgenden Kapiteln anzuwenden.
Ein kleiner Zwischenstand in Kapitel 3.8 wird diesen Abschnitt abschließen.

3.1 Lernmethoden

Es existieren unterschiedliche Konzepte zum Thema Lernen. Diese Konzepte können auch in der Gesellschaft der Menschen beobachtet werden. Ein Mensch ist von Geburt an lernfähig. Er erhält die Möglichkeit zum Beispiel aus gemachten Fehlern zu lernen. Dies ist die langsamere Variante. Der Mensch kann auch aus Erfahrungen der Anderen (Lehrer) lernen. Solch ein Lernprozess ist schneller, aber von den Trainingsdaten, also der Erfahrung der Lehrer, abhängig. Ein Mensch kann von seinen Vorfahren bestimmte Fähigkeiten erhalten haben, also genetisch bedingte Talente besitzen. Er hat auch die Fähigkeit bestimmte Erfahrungen zu gruppieren, vorgegebene Daten zu vervollständigen und Verknüpfungen (Assoziationen) mit existierendem Wissen zu bilden. Dies erfolgt ohne die richtigen Ergebnisse zu kennen, ohne zu wissen, ob gewisse Zusammenhänge richtig sind. Alle drei Konzepte haben im Bereich des maschinellen Lernen drei besondere Namen: bestärkendes Lernen (engl.: reinforcement learning), überwachtes Lernen (engl.: supervised learning) und unüberwachtes Lernen (engl.: unsupervised learning). Alle drei Konzepte haben eine berechtigte Existenz.
Das Konzept des überwachten Lernen beinhaltet zum größten Teil die Lehre über neuronale Netze. Ein breites Spektrum von Arten der neuronalen Netze mit ihren Lernmethoden und vielen Anwendungsbeispielen ist z. B. in [Zell, 2003] beschrieben. Eine immer größere Bedeutung gewinnt in letzter Zeit das Konzept der Support-Vector-Machine (SVM) [Vapnik, 1998], das für die Klassifikationsaufgaben und für die Regression angewendet werden kann.
Das Konzept des unüberwachten Lernen, ist auch unter dem Begriff Data Mining bekannt. Bei dieser Art des Lernens sollen die Regelmäßigkeiten aus der Menge der Trainingsdaten extrahiert werden. Beispiele für unüberwachtes Lernen bilden die Clustering-Methoden [Fisher, 1987] sowie Erkenntnisgewinnung (engl.: knowledge discovery) [Langley, 2001] oder auch selbstorganisierende Karten [Kohonen, 1982].
In der vorliegenden Arbeit wird aber die größte Aufmerksamkeit dem Konzept des bestärkenden Lernen [Sutton & Barto, 1998] gewidmet. Dieses Lernkonzept liegt zwischen überwachtem und unüberwachtem Lernen. Es liegen keine Trainingsdaten vor, es existiert aber ein rudimentäres Signal, später auch Belohnung (engl.: reward) genannt. Dieses Signal beinhaltet eine Information, ob die durchgeführte Aktion schlecht, gut oder neutral war. Dieses Konzept basiert auf der Idee aus den elementaren Belohnungen eine Gesamtbelohnung zusammenzustellen, die eine Vorstellung über die Güte der gesamten Kette der durchgeführten Aktionen bis zum Erreichen des Ziels besitzt. Das sogenannte Versuch-und-Irrtum-Prinzip (engl.: trial and error) steht im Ursprung dieser Methode. Dies entspricht dem Naturprinzip über Schlüsselreize, vergleiche auch [Mazur, 2006].

3.2 Reinforcement-Learning

Reinforcement-Learning ist ein wichtiger Bereich auf dem Gebiet des maschinellen Lernens [Heidrich-Meisner et al., 2007]. Es entsteht durch das Lernen aus der Interaktion mit der Umgebung. In den dargestellten Lernalgorithmen wird davon ausgegangen, dass alle Agentenarten den aktuellen Zustand sicher erfassen können.

Der Vorteil dieses Verfahrens liegt darin, dass ohne ein Weltmodell gelernt werden kann. Dabei kann das Lernen direkt im Onlinebetrieb erfolgen. Die schon gewonnenen Erkenntnisse können direkt in die Entscheidungsfindung einbezogen werden. Die Möglichkeit das Wissen im Onlinebetrieb zu aktualisieren hat insbesondere für Navigationsprobleme eine große Bedeutung. Ein Navigationsproblem bereitet folgende Schwierigkeiten: Messungen können erst während des Betriebs ergänzt bzw. ermittelt werden, einige Informationen können erst dann bestimmt werden, wenn andere Daten vorliegen und können auch erst dann bei der Lösung des Problems berücksichtigt werden, also sind die benötigten Daten nicht vollständig.
Die Verfahren des RL benötigen keine aufwendige Erstellung einer Trainingsmenge. Es genügt, ein rudimentäres Belohnungssignal zu definieren, das nur die lokale Auswirkung der Aktionen bewerten muss, was relativ einfach ist. Der RL-Algorithmus extrahiert aus lokalen, durch ein Belohnungssignal repräsentierten Belohnungswerten den globalen Zusammenhang der einzelnen Handlungen in Bezug auf das verfolgte globale Ziel in Form einer Nutzenfunktion, die für die einzelnen Zustände die Gesamtbelohnung repräsentiert. Das Ziel beim Lernprozess ist es die optimale Nutzenfunktion zu erlernen. Aus der optimalen Nutzenfunktion kann die optimale Strategie erstellt werden, die eine optimale Folge von Zustand-Aktions-Paaren in Bezug auf das globale Ziel darstellt.
Die Reinforcement-Learning-Methoden werden erfolgreich für zahlreiche Steuerungs- und Entscheidungsprobleme eingesetzt [Heidrich-Meisner et al., 2007], zum Beispiel optimale Steuerungsprobleme, Zeitplanoptimierung, Spieltheorie (für zwei oder mehr Spieler) [Hu & Wellman, 1998], in Multiagentensystemen beim RoboCup [Stone et al., 2005], u.a..

3.2.1 Markow-Entscheidungsproblem

"[Die] Welt [ist] als ein dynamisches System modelliert, dessen Entwicklung diskret in der Zeit voranschreitet und von getroffenen Entscheidungen vorangetrieben wird." [Jung, 2007, Kap. 1.1, S. 2]. In jedem Zeitschritt befindet sich das System in einem Zustand, und geht unter Auswahl einer Aktion in einen neuen Zustand über.
Für Markow-Entscheidungsprobleme, bei denen der Nutzen eines Agenten von einer Folge von Entscheidungen abhängt, gilt die Markow'sche Annahme. Die Markow'sche Annahme bedeutet, dass die Wahrscheinlichkeit dafür, dass ein Zustand s_{t+1} mit der dafür erhaltenen Belohnung r_t, der von einem Zustand s_t ausgehend durch die Aktion a_t zu erreichen ist, nur von s_t und der durchgeführten Aktion a_t und nicht von den Vorgängern von s_t und a_t abhängt, also wenn für alle s_{t+1}, r_t, s_t und a_t die folgende Gleichung gilt.

$$P(s_{t+1} = s', r_t = r|s_t, a_t) = P(s_{t+1} = s', r_t = r|s_t, a_t, r_t, \ldots, r_1, s_0, a_0)$$

Die Transitionswahrscheinlichkeit ist wie folgt vorgegeben:

$$P^a_{ss'} := P\{s_{t+1} = s'|s_t = s, a_t = a\} = P\{s_{t+1} = s'|s_t = s, a_t = a, \ldots, s_0, a_0\}$$

Die Entscheidung über die nächste auszuführende Aktion a_t wird nur auf Grundlage des aktuellen Zustands s_t getroffen. Der Zustand s_{t+1} wird von Zustand s_t durch Ausführen der Aktion a_t erreicht. Alle für die Zukunft relevanten Informationen (ggf. auch Informationen aus der Vergangenheit) sind im aktuellen Zustand ohne die Betrachtung der vergangenen kompletten Folge der Zustand-Aktions-Paare enthalten.
"Jede Auswahl einer Aktion ist mit einem reellwertigen Nutzen (Belohnung, engl.: reward) verbunden,

der von s_t, a_t und dem eingetretenen Folgezustand s_{t+1} abhängt. Die Belohnung ist extern vorgegeben und bewertet nur das unmittelbare Eintreten eines Ereignisses." [Jung, 2008]

$$R^a_{ss'} := E\{r_t | s_{t+1} = s', s_t = s, a_t = a\} = E\{r_t | s_{t+1} = s', s_t = s, a_t = a, ..., s_0, a_0\}$$

$R^a_{ss'}$ ist die unmittelbare Belohnung (engl.: immediate reward). E steht für Erwartungswert.
Ein Markow-Entscheidungsproblem ist ein Tupel $(S, A, P^a_{ss'}, R^a_{ss'})$, wobei S die endliche Menge der Zustände ist. A ist die endliche Menge der Aktionen.
Das Markow-Entscheidungsproblem wird für endliche Zustands- und Aktionsräume formuliert. Die betrachteten Szenarien erfüllen diese Bedingung nicht. "Hierbei wird mittels Regression aus Stichproben eine Funktion bestimmt, die die Lösung einer *Optimalitätsgleichung* (Bellman) ist, und aus der sich näherungsweise optimale Entscheidungen ableiten lassen. Zentrales Problem ist dabei die Dimensionalität des Zustandsraums, die häufig hoch und daher traditionellen gitterbasierten Approximationsverfahren wenig zugänglich ist." [Jung, 2008]

3.2.2 Nutzenfunktion

Die vom Agenten gesammelte Erfahrung muss in einer geeigneten Form dargestellt bzw. gespeichert werden. Diese Erfahrung soll nicht nur auf der elementaren Belohnung beruhen, sondern auf der Gesamtbelohnung, die von der Kette der gemachten Entscheidungen über die Aktionen abhängt, wenn der Agent vom Zustand s_0 startet. Diese Kette der gemachten Entscheidungen wird als *Strategie* π bezeichnet.
Die Strategie π (engl.: policy) wird in dieser Arbeit als Abbildung $\pi : S \to A$ definiert. D.h. die Strategie schreibt jedem Zustand $s \in S$ eine Aktion $\pi(s) =: a \in A$ vor. Mit der Strategie wird die Nutzenfunktion assoziiert, die für jeden Zustand eine erwartete Gesamtbelohnung (in Bezug auf das globale Ziel) darstellt. Es gilt auch das Kernprinzip, dass aus einer vorgegebenen Nutzenfunktion eine Strategie abgeleitet werden kann.
Die Definition der Transitionsfunktion $\tau(s, a)$, siehe Gleichung 3.1, die vom Zustand s_t und nach dem Ausführen der mit der Strategie π vorgeschlagenen Aktion den nächsten Zustand des Agenten liefert, wird in dieser Arbeit wie folgt eingeführt:

$$s_{t+1} =: \tau(s_t, \underbrace{\pi(s_t)}_{a_t}) = \max_{s'} P^a_{ss'} \tag{3.1}$$

Die Nutzenfunktion kann z. B. als gewichtete Summe der elementaren Belohnungen für jeden Zustand des Zustandsraums und die verfolgte Strategie π definiert werden. Die Nutzenfunktion kann als eine V- oder eine Q-Funktion[12] eingeführt werden. Die V-Funktion bildet jeden Zustand $s \in S$ auf seine Gesamtbelohnung ab, die aus den einzelnen Belohnungen mit der verfolgten Strategie π und dem Startzustand s gebildet wird, siehe Gleichung (3.2). Für diese Art der Repräsentation der Nutzenfunktion wird vom Agenten Wissen über sein Transitionsmodell benötigt.

$$V^\pi(s) = E_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s) \tag{3.2}$$

Die in Gleichung (3.3) definierte Q-Funktion beinhaltet die Informationen über den Gesamtnutzen aller Zustand-Aktions-Paare (s, a), $s \in S, a \in A$ und kann auch in Systeme eingebunden werden, wo das

[12] Diese Funktionen werden als Tabellen für das endliche MDP realisiert.

Transitionsmodell unbekannt ist oder die Auswirkung der einzelnen Aktionen nicht immer vorhersehbar ist.

$$Q^{\pi}(s,a) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a) \tag{3.3}$$

Die V-Funktion bzw. Q-Funktion enthält eine rekursive Verknüpfung. Diese Verknüpfung ist in der Bellman'schen Gleichung (3.4) für V^{π} dargestellt. Die in der Definition der V-Funktion eingebundene Rekursion ist eine fundamentale Eigenschaft für die Theorie der dynamischen Programmierung bzw. des Reinforcement-Learnings [Sutton & Barto, 1998, Kap. 3.7 S. 70].

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s) \\ &= E_{\pi}(r_t + \gamma \cdot \underbrace{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}}_{=V^{\pi}(s')} | s_t = s) \\ &\overbrace{=}^{\text{stochastische } \pi} \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi}(s') \right] \\ &\overbrace{=}^{\text{deterministische } \pi} \sum_{s'} P^{\pi(s)}_{ss'} \left[R^{\pi(s)}_{ss'} + \gamma V^{\pi}(s') \right] \end{aligned} \tag{3.4}$$

wobei $\pi(s,a)$ die Wahrscheinlichkeit für das Durchführen der Aktion a im Zustand s unter der stochastischen Strategie π ist. Wenn die Strategie π deterministisch ist, kann dieser Wert wie folgt definiert werden: $\pi(s,a) = \{1, \text{wenn Aktion } a \text{ gewählt ist, sonst } 0\}$.

3.2.3 Optimalität

Die V-Funktion bzw. Q-Funktion bestimmt die Ordnungsrelation der möglichen Strategien. D. h. mithilfe der V- bzw. Q-Funktion kann eine Strategie π in Relation zu einer anderen Strategie π' gestellt werden. Eine Strategie π ist besser oder gleich gut wie eine andere Strategie π', wenn für jeden $s \in S$ gilt $V^{\pi}(s) \geq V^{\pi'}(s)$ bzw. $Q^{\pi}(s,\pi(s)) \geq Q^{\pi'}(s,\pi'(s))$ [Sutton & Barto, 1998, Kap. 3.8, S. 75].
Mithilfe dieser Ordnungsrelation kann die optimale Strategie, die durch die optimale V^*-Funktion repräsentiert wird, eingeführt werden. Die Strategie ist dann optimal, wenn alle anderen Strategien schlechter sind, also $V^*(s) := V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \forall s \in S$.
Die optimale Strategie ist über die Q-Funktion wie folgt definiert:

$$Q^*(s,a) := Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \forall s \in S \text{ und } a \in A(s).$$

Zwischen V^*- und Q^*-Funktionen gilt die folgende Verbindung für eine gierige Strategie: $V^*(s) = \max_{a \in A(s)} Q^{\pi^*}(s,a)$, vergleiche [Sutton & Barto, 1998, Kap. 3.8, S. 76].
Die Grundidee aller RL-Verfahren liegt in der Anwendung der Nutzenfunktion zur Suche nach optimalen Strategien. Dieser Suchprozess heißt im Englischen Generalized Policy Iteration (GPI) und besteht aus zwei alternierenden, interaktiven Prozessen: Strategie-Auswertung und Strategie-Verbesserung [Sutton & Barto, 1998, Kap. 4.6, S. 105]. Ausgangspunkt der GPI ist eine Nutzenfunktion sowie eine beliebige Anfangsstrategie π.

Das Policy-Improvement-Theorem [Sutton & Barto, 1998, Kap. 4.2, S. 93 ff] lautet: Es existieren zwei Strategien π und π'. Wenn für jeden $s \in S$ die folgende Ungleichung gilt,

$$Q^{\pi'}(s, \pi'(s)) \geq V^{\pi}(s) \tag{3.5}$$

dann ist die Strategie π' besser als oder gleich gut wie die Strategie π.
Eine sogenannte gierige Strategie, siehe Gleichung (3.6), erfüllt die Bedingung (3.5), wenn die Q^{π}- bzw. V^{π}-Funktion die optimale Nutzenfunktion repräsentiert.

$$\pi^*(s) := argmax_a V^*(\tau(s,a)) = argmax_a Q^*(s,a) \tag{3.6}$$

Durch die Anwendung der gierigen Strategie π auf die aktuelle Nutzenfunktionen V bzw. Q im aktuellen Zustand s_t wird die aktuelle Aktion bestimmt (Verbesserungsschritt).
Die Ausführung dieser Aktion führt zum Schritt der *Auswertung*. Die neuen Messungen werden mit alten Werten der Nutzenfunktion verglichen (TD-Fehler), und auf dieser Grundlage wird die Nutzenfunktion V^{neu}- bzw. Q^{neu} ausgewertet.
Anschließend findet wieder die *Verbesserung* der Strategie statt: Ausgehend von der neuen Nutzenfunktion V^{neu}- bzw. Q^{neu} wird mit der gierigen Strategie π' eine neue Aktion bestimmt. Diese Strategie π' kann sich von der Strategie π unterscheiden, da die Nutzenfunktion V^{neu} bzw. Q^{neu} aktualisiert ist.
So entsteht ein Zyklus. Die Ausführung dieses Zyklus führt zur Annäherung der Nutzenfunktion an die optimale Nutzenfunktion. Aus der optimalen Nutzenfunktion kann durch die gierige Strategie die gesuchte optimale Strategie bestimmt werden.
Die Gleichung (3.4) kann für die gierige deterministische Strategie, siehe Gleichung (3.6), in die Bellman'sche Optimalitätsgleichung umgestellt werden:

$$V^*(s) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(\tau(s,a))\right] \tag{3.7}$$

Für die gierige Strategie π^* ist die Bellman'sche Optimalitätsgleichung für die Q-Funktion wie folgt definiert:

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(\tau(s,a), a')\right] \tag{3.8}$$

Wenn in der Aufgabe alle Zustände $s \in S$ sowie alle möglichen Übergänge $R^a_{ss'}$ für die in den Zuständen s möglichen Aktionen $A(s)$, also die Dynamik der Umgebung, bekannt sind, und die Anzahl der Zustände klein ist, also die Ressourcen in der Lage sind das aufgestellte Gleichungssystem aufzulösen, und die Markow-Eigenschaft erfüllt ist, kann eine Auflösung des Gleichungssystems (3.7) bzw. (3.8) die optimale V- bzw. Q-Funktion und daraus die optimale Strategie erzielen, siehe [Sutton & Barto, 1998, Kap. 3.8, S. 79]. In vielen Fällen ist es aber nicht nötig die Lösung für alle Zustände zu bestimmen. Wenn praxisorientierte Anwendungen beabsichtigt werden sollen, ist ein genaues Umgebungsmodell meistens nicht bekannt. Es existiert nur die Möglichkeit die Veränderungen in den einzelnen Schritten zu messen. Somit kann das Gleichungssystem (3.7) bzw. (3.8) nicht immer aufgestellt bzw. gelöst werden. Durch den Ansatz der iterativen Approximation wird die sukzessive Annäherung der Nutzenfunktion realisiert. Für diese Fälle werden Algorithmen entwickelt, die einzelne Werte der V- bzw. Q-Funktion in Abhängigkeit von den aktuellen Messungen aktualisieren. Und so kann eine Approximation der optimalen Lösung der Bellman'schen Gleichung erzielt werden. Der SARSA-Algorithmus realisiert die oben beschriebene Idee [Sutton & Barto, 1998, Kap. 3.9, S. 80].

3.3 SARSA-Algorithmus im diskreten Zustandsraum

Während des Lernprozesses im Onlinebetrieb soll das Lernen und die Anwendung des Gelernten bei der Entscheidung über die nächste Aktion gleichzeitig ablaufen. Der SARSA-Algorithmus basiert auf beiden Schritten der GPI. In die Entscheidung fließt der aktuelle Wissensstand über die Umgebung sowie die Ziele ein. Die beiden Prozesse laufen mithilfe der Aktualisierungs- und Anwendungsschritte der Nutzenfunktion ab.
Der verwendete SARSA-Algorithmus ist eine Realisierung des Reinforcement-Learning-Ansatzes, der Lernen im Onlinebetrieb erlaubt. Der Agent lernt im endlichen MDP. Der SARSA-Algorithmus wurde von Rummery und Niranjan [Rummery & Niranjan, 1994] eingeführt, wobei es als modifiziertes Q-Lernen bezeichnet wurde.
In [Singh et al., 2000] wurde gezeigt, dass der SARSA(0)-Algorithmus unter der Verwendung der "GLIE learning policies"[13] zur optimalen Q^*-Funktion, also zur optimalen Strategie unter bestimmten Annahmen für den Lernschritt α und für die Wahl der elementaren Belohnungen r, konvergiert. Ein Beispiel für die gewünschte GLIE-Strategie haben die Autoren als ϵ-gierige Strategie mit ϵ-Anteil als eine Funktion $\epsilon_t(s) = \frac{c}{n_t}$ genannt. Die Konstante $0 < c < 1$ in dieser Definition der GLIE-Strategie ist im Vorfeld vorgegeben, $n_t(s) \leq t$ gibt an, wie oft der Zustand s in t Zeitschritten besucht wurde.

3.3.1 Herleitung der Update-Regel

Die Interaktion des rationalen Agenten in der Umgebung ist ein dynamisches System, das durch ein endliches Markow-Entscheidungsproblem formuliert werden kann. Dabei befindet sich der Agent in jedem diskreten Zeitschritt $t = 0, 1, 2, \ldots$ im Zustand $s_t \in S$ und wählt eine Aktion a_t aus, gemäß gieriger Strategie an die aktuelle Q-Funktion im Zustand s_t angewendet. Wenn die Q-Funktion noch nicht die gesuchte tatsächliche Nutzenfunktion ist, wird sie mithilfe von GPI an die gesuchte optimale Nutzenfunktion angenähert.
In jedem Zeitschritt t wird der Temporal-Difference-Fehler δ_t (TD-Fehler) für den aktuellen $Q(s_t, a_t)$-Wert im aktuellen Zustand s_t und die Aktion a_t gebildet. Dieser Temporal-Difference-Fehler wird als Differenz zwischen der aktuellen Messung ($r_t + \gamma Q(s_{t+1}, a_{t+1})$) und dem aktuellen $Q(s_t, a_t)$-Wert definiert, siehe Gleichung (3.9). Wobei der Folgezustand nach dem Transitionsschritt bestimmt wird: $s_{t+1} \overset{(3.1)}{\leftarrow} \tau(s_t, a_t)$, die neue Aktion $a_{t+1} \leftarrow \max_a(Q(s_{t+1}, \cdot))$ wird gemäß der gierigen Strategie bestimmt. Die alte Schätzung der Gesamtbelohnung für den aktuellen Zustand s_t und die aktuelle Entscheidung für die Aktion a_t ist im $Q(s_t, a_t)$-Wert versteckt.

$$\epsilon(Q(s_t, a_t)) := \frac{1}{2}\delta_t^2 = \frac{1}{2}(r_t + \gamma \cdot Q(\underbrace{s_{t+1}}_{=\tau(s_t,a_t)}, a_{t+1}) - Q(s_t, a_t))^2 \tag{3.9}$$

Mit der Anwendung der stochastischen Gradientenverfahren zur Korrektur des aktuellen Eintrags in der Q-Funktion entsteht die Update-Regel für den SARSA-Algorithmus, siehe Gleichung (3.10).

$$\begin{aligned} Q(s_t, a_t)^{neu} &\leftarrow Q(s_t, a_t) - \frac{\partial \epsilon(Q(s,a))}{\partial Q(s,a)}|_{Q(s_t,a_t)} \\ &\leftarrow Q(s_t, a_t) + \alpha \cdot \underbrace{(\underbrace{r_t + \gamma \cdot \max_a(Q(\tau(s_t, a_t), a))}_{\text{neue Schätzung}} - \underbrace{Q(s_t, a_t)}_{\text{alte Schätzung}})}_{\delta_t} \end{aligned} \tag{3.10}$$

[13] Die Bezeichnung GLIE steht für "Greedy in the Limit with Infinite Exploration"

Da die Update-Regel jedes Element aus dem Quintupel $(s_t, a_t, r_t, \underbrace{s_{t+1}}_{=\tau(s_t,a_t)}, \underbrace{a_{t+1}}_{=\max_a(Q(s_{t+1},\cdot))})$ nutzt, heißt das Verfahren SARSA-Algorithmus.
Die Update-Regel ist für die gierige Strategie an Stelle der Bestimmung der neuen Aktion (Handlung) formuliert. Die neue Aktion kann auch mit einem gewissen Zufallsfaktor zur Untersuchung weiterer Möglichkeiten als zufällige Aktion ausgewählt werden, so dass $\max_a Q(\tau(s_t, a_t), a)$ mit einer gewissen Wahrscheinlichkeit, an dieser Stelle als ϵ bezeichnet, gegen eine zufällige Aktion ausgetauscht werden kann. Diese Strategie, die mit der Wahrscheinlichkeit $(1 - \epsilon)$ die gierige Wahl der Aktion trifft und mit der Wahrscheinlichkeit ϵ die zufällige Wahl der nächsten Aktion trifft, wird als ϵ-gierige Strategie bezeichnet. Die ϵ-gierige Strategie ist eine Näherung der GLIE-Strategie. Durch den ϵ-Anteil wird der Explorationsschritt simuliert, vergleiche Problemgenerator im lernenden Agenten in Abbildung 2.2.

3.3.2 SARSA-Algorithmus im diskreten Fall

Der SARSA-Algorithmus wird zuerst für das Markow-Entscheidungsproblem mit endlichen Zustands- und Aktionsräumen eingeführt. Der große Vorteil dieses Verfahrens liegt in der Möglichkeit während des Ausführens der Episode auf die gesammelten Erfahrungen zurückzugreifen, da die Q-Funktion sofort korrigiert wird, und die korrigierte Q-Funktion direkt angewendet wird. Der Nachteil liegt in der schwachen Konvergenz, also im langsamen Lernen. Die Update-Regel (3.10) und die ϵ-gierige Strategie bilden den Kern des SARSA-Algorithmus. Im Algorithmus 3.1 wird die Realisierung des SARSA-Algorithmus präsentiert. In Zeile 9 wird die neue Aktion aus der aktuellen Q-Funktion gemäß der ϵ-gierigen Strategie bestimmt. Die Bestimmung der neuen Aktion ist im Algorithmus als eine Funktion realisiert, vergleiche Zeile 14. Die neu bestimmte Aktion wird im nächsten Zeitabschnitt in Zeile 8 ausgeführt. In Zeile 10 wird die Q-Funktion für das aktuelle Zustand-Aktions-Paar an die aktuellen Messungen angepasst.
Die konstanten Werte α, γ, vergleiche Zeile 10, und ϵ, vergleiche Zeile 16, sind vorgegeben.

Algorithmus 3.1 SARSA-Algorithmus

```
 1: procedure SARSA
 2:     Initialisiere Q-Funktion
 3:     repeat Für jede Episode j                                   ▷ j = 0, ..., E
 4:         t ← 0
 5:         Initialisiere s_0                  ▷ Startposition s_0 ist in der Regel gleich
 6:         a_0 ← ε − gierig(s_0)              ▷ Bestimme a_0 gemäß ε-gieriger Strategie
 7:         repeat Für jeden Schritt t der Episode j                ▷ t = 0, ... T
 8:             Führe die Aktion a_t aus, beobachte r_t, s_{t+1}
 9:             a_{t+1} ← ε − gierig(s_{t+1})   ▷ Bestimme a_{t+1} gemäß ε-gieriger Strategie
10:             Q(s_t, a_t) ← Q(s_t, a_t) + α(r_t + γ · Q(s_{t+1}, a_{t+1}) − Q(s_t, a_t))
11:             t ← t + 1
12:         until Bis s_t der Zielzustand ist
13:     until
14: function ε − gierig(s)
15:     r ← Zufall(0, 1)                   ▷ Wähle Zufallszahl aus dem Intervall [0, 1]
16:     if r > ε then
17:         return a ← max_a Q(s, a)
18:     else return a ← Zufall(A(s))
```

3.3.3 Bedeutung der Lernparameter α, ϵ, γ, λ

Der Diskontierungsfaktor $0 < \gamma \leq 1$ ist eine Gewichtung der Bedeutung einer elementaren Belohnung in der Zukunft. Mithilfe des Diskontierungsfaktors wird die Gesamtbelohnung für die einzelnen Ketten der Entscheidungen $\sum_{t=0}^{T} \gamma^t r_t = r_0 + \gamma \cdot r_1 + \gamma^2 \cdot r_2 + ...$ gebildet, auf dieser Grundlage wird die Nutzenfunktion definiert, vergleiche Gleichung (3.2) bzw. (3.3). Die Gewichtung γ^t der einzelnen elementaren Belohnungen aus der Kette der Entscheidungen wird in Abhängigkeit vom Zeitschritt t definiert. Je weiter eine Belohnung r_t in der Zukunft liegt, also je größer der Zeitschritt t ist, desto kleiner ist der Einfluss dieser Belohnung in der geschätzten Gesamtbelohnung. Dies wird mit dem Wert des Parameters γ gesteuert. Je kleiner γ gewählt wird, desto kleiner ist die Anzahl der Schritte, die die Gesamtbelohnung beeinflussen. Folgendes Beispiel zeigt den Einfluss von zwei unterschiedlichen Werten für γ für den 6. Schritt. Mit $\gamma = 0.1$ fließt die elementare Belohnung r_5 mit einer Gewichtung von $0.1^5 = 0.00001$ in die Summe ein. Dieser Wert hat kaum noch einen Einfluss in der Berechnung der Summe. Für $\gamma = 0.99$ wird die Gewichtung auf $0.99^5 = 0.9501$ gesetzt. Dieses Gewicht ist relativ hoch und hat noch fast den vollen Einfluss in der Berechnung der Summe. Wenn der Wert für den Parameter γ auf einen kleinen Wert gesetzt wird, ist der Agent "kurzsichtig". Hingegen, wenn der Parameter γ auf einen Wert nahe Eins gesetzt wird, wird der Agent "vorausschauend" genannt, siehe [Baird et al., 1999].
Der Parameter $0 < \alpha \leq 1$ bestimmt die Lernschrittweite für das Gradientenabstiegsverfahren. Die Lernschrittweite bestimmt, wie viel von der neuen Erfahrung in das gegenwärtige Wissen einbezogen wird. Ein sehr kleines α verlangsamt den Lernprozess, ein sehr hohes α hingegen birgt die Gefahr, dass die Schlucht der Fehlerfläche, in der ein gutes Minimum liegt, übersprungen wird. Somit können die Lernerfolge mit fortschreitender Lernzeit wieder verloren gehen oder sich eventuell gar nicht erst einstellen. Oft wird empfohlen, mit relativ großer Schrittweite zu beginnen und diese langsam zu vermindern. Für einfache Probleme reicht es oft aus, einen konstanten Wert für die Schrittweite beizubehalten [Zell, 2003, Kap. 8, S. 114].
Für den Parameter λ, den Abklingfaktor (engl.: decay factor), gilt: $0 \leq \lambda \leq 1$. Mit wachsendem λ wächst auch der Einfluss der neuen Erfahrung auf die vorangegangenen Zustand-Aktions-Paare unter der Anwendung der Eligibility-Traces. Bei hohen Werten für den Parameter λ wird von langen Spuren gesprochen, da für die meisten besuchten Zustand-Aktions-Paare der Einfluss der neuen Erfahrung tatsächlich sehr hoch ist. Wenn $\lambda = 0$ bestimmt wird, wird die Länge der Spuren der Eligibility-Traces gleich Null. $\lambda = 0$ entspräche der Aktualisierung der Nutzenfunktion gemäß der Update-Regel (3.10). Für $\lambda = 1$ fällt der Eligibility-Trace nur mit γ ab. Bei $\gamma = 1$ kombiniert mit $\lambda = 1$ werden die Eligibility-Traces nie herabgesetzt. Dieser Parameter wirkt sich sehr stark auf die Lerngeschwindigkeit aus. Im MCP wird dieser Parameter variiert. Für das untersuchte Navigationsproblem wird dieser Wert auf einen konstanten Wert gesetzt.
In [Singh & Sutton, 1996, S. 23] wird von einem sehr großen $\lambda > 0.99$ bzw. der Wahl von $\lambda = 0$ in Kombination mit der Tile-Coding-Methode abgeraten, da sich die Leistung in beiden Fällen rapide verschlechtern kann. In [Singh & Sutton, 1996, S. 17, 22] werden mittlere Werte für die Parameter α und λ empfohlen.
Für manche Anwendungen und Versuchsreihen ist es interessant den Lernprozess ohne Einfluss der Eligibility-Traces zu untersuchen, also λ auf Null zu setzen. In diesem Fall hat sich ein Parameter von $\alpha = 0.5$ als gut erwiesen.
Die Standardparameter für den SARSA-Algorithmus im Navigationsproblem wurden für einen diskreten und kontinuierlichen Zustandsraum in einer Diplomarbeit [Wolter, 2007] bestimmt. Die Ergebnisse werden in der vorliegenden Arbeit angewendet. Die Standardwerte für die Parameter im Navigationsproblem sind wie folgt gesetzt: ($\alpha = 0.2, \epsilon = 0.1, \lambda = 0.8$). Der Parameter γ wird in vielen Fällen variiert und in

Verbindung mit den durch die Erweiterung entstandenen zusätzlichen Parametern bestimmt.
Für das MCP wird in [Taylor et al., 2008] eine Parametrisierung von $\alpha = 0.5, \epsilon = 0.1, \lambda = 0.95$ als die beste Lösung genannt. Das Kriterium für die beste Lösung war die Bestimmung der Strategie mit minimaler Anzahl der bis zum Ziel benötigten Schritte. In [Sutton & Barto, 1998] wird für ein MCP eine Parametrisierung von $\lambda = 0.9, \epsilon = 0, \alpha = 0.05$, Tile-Coding-Methode mit einer Zerlegung von 11×11 Teilen und zwei Tilings angewendet. In der vorliegenden Arbeit werden die Parameter zu $\alpha = 0.2$, $\epsilon = 0$ gesetzt. Die Parameter λ und γ werden variiert und die Tile-Coding-Methode mit einer Zerlegung von 11×11 Teilen und einem Tiling angewendet.

3.4 SARSA-Algorithmus im kontinuierlichen Zustandsraum

An dieser Stelle wird die Anwendung des SARSA-Algorithmus 3.1 für die Aufgabe mit stetigem Zustandsraum präsentiert. In Unterkapitel 3.4.1 wird zuerst die Notwendigkeit der Diskretisierung des Zustandsraumes und die Notwendigkeit der Approximation der Nutzenfunktion begründet. In Unterkapitel 3.4.2 wird die allgemeine lineare Approximation der Nutzenfunktion eingeführt. In Unterkapitel 3.4.3 werden die verschiedenen Möglichkeiten den Zustandsraum zu diskretisieren eingeführt. Das Unterkapitel 3.4.4 führt die Möglichkeit der Diskretisierung des Zustandsraumes mithilfe radialer Basisfunktionen und die Approximation der Nutzenfunktion mit einem radialen Basisfunktionen-Netzwerk (RBF-Netzwerk) ein. Die mögliche Erweiterung des Lernens wird mit der Approximation der Nutzenfunktion mithilfe eines RBF-Netzwerks mit theoretischen Hintergründen und Herleitungsschritten in Unterkapitel 3.4.5 präsentiert. Der erweiterte Algorithmus ist in Unterkapitel 3.4.6 zusammengefasst.

3.4.1 Motivation

Wenn die Anzahl der Zustände im Zustandsraum unendlich groß ist, z. B. die Zustände nicht nur diskrete Werte aus dem ganzen Intervall annehmen können, wird in dieser Arbeit von einem kontinuierlichen Zustandsraum gesprochen. Die in dieser Arbeit betrachteten Probleme, das Mountain-Car-Problem und das Navigationsproblem besitzen einen kontinuierlichen Zustandsraum.
Für kontinuierliche Zustandsräume ist es nicht mehr möglich, jeden Zustand s ein- oder mehrfach zu besuchen, wie es noch für diskrete Zustandsräume möglich war, und es laut Konvergenzbedingung für den SARSA-Algorithmus 3.1 benötigt wird. Für kontinuierliche Zustandsräume muss der Agent generalisieren, d.h. er muss aufgrund seiner beschränkten Erfahrung Rückschlüsse bezüglich unbekannter Zustände und Aktionen ziehen. Für Algorithmen, die auf der Auswertung und Entwicklung von Nutzenfunktionen basieren, bedeutet das, dass die Nutzenfunktion approximiert werden muss [Röttger, 2009].
Durch den Diskretisierungsschritt wird die tatsächliche Nutzenfunktion, die für den echten kontinuierlichen Zustandsraum definiert ist, durch eine Approximation ersetzt. In [Perkins & Precup, 2002] und [Melo et al., 2008] wurde gezeigt, dass diese Annäherung der Q-Funktion mit einer linearen Funktion unter der Voraussetzung einer geeigneten Diskretisierung und Anwendung des SARSA-Algorithmus zu einer guten Lösung konvergiert.

3.4.2 Lineare Funktionsapproximation

In der vorliegenden Arbeit wird die Nutzenfunktion durch eine Funktion mit linearer Kombination des Parametervektors und Basisvektors approximiert. Die lineare Funktionsapproximation der Nutzenfunktion wird eingesetzt. In [Tsitsiklis & Roy, 1996] wurde gezeigt, dass die nichtlineare Funktionsapproximation zur Divergenz der Lernalgorithmen führen kann. Die Diskretisierung des Zustandsraumes wird mit-

hilfe einer endlichen Basis von Funktionen $\{\phi_1(), \dots \phi_N()\}$, die lokalisierende Eigenschaften aufweisen, realisiert. Ein Beispiel für solche Funktionen ist die Impulsfunktion oder die radiale Basisfunktion. Notwendige Bedingungen und der Aufbau einer solchen Basis von Funktionen werden in den nachfolgenden Kapiteln beschrieben. Die endliche Menge der Basisfunktionen bildet einen Basisvektor, wobei jede Funktion aus der Basis einen Eintrag im Basisvektor $(\phi_1(), \dots, \phi_N())^T$ hat. Eine Basisfunktion aus einer endlichen Basis wird auch als Merkmal bezeichnet. Jeder Zustand s aus dem kontinuierlichen Zustandsraum S wird durch einen solchen Basisvektor $(\phi_1(s), \dots, \phi_N(s))^T$ repräsentiert. Der Gewichtsbasisvektor wird aus dem Basisvektor gebildet. Mit dem Gewichtsmerkmalvektor werden die Werte aus dem Parametervektor je nach Wichtigkeit der einzelnen Merkmale für den aktuellen Zustand s gewichtet[14], siehe Gleichung (3.11).

$$\mathbf{w(s)} = \frac{1}{\sum_i \phi_i(s)} (\phi_1(s), \dots \phi_N(s))^T \tag{3.11}$$

Die lineare Funktionsapproximation der Nutzenfunktion ist als Skalarprodukt des Gewichtsbasisvektors und des Parametervektors θ in der Gleichung (3.12) bzw. für die bestimmte Aktion a des Anteils des Parametervektors $\theta_{\mathbf{a}}$ in Gleichung (3.13) dargestellt. Die Anzahl der Einträge im Parametervektor θ für die V-Funktion bzw. für den Anteil des Parametervektors $\theta_{\mathbf{a}}$ der Aktion $a \in A$ für die Q-Funktion ist gleich der Anzahl der Funktionen in der ausgewählten Basis $(\phi_1(), \dots, \phi_N())$.

$$V_\theta(s) = \theta^T \cdot \mathbf{w(s)} = \frac{1}{\sum_i \phi_i(s)} \sum_{i=1}^{N} \theta(i) \cdot \phi_i(s) \tag{3.12}$$

$$Q_\theta(s,a) = {\theta_{\mathbf{a}}}^T \cdot \mathbf{w(s)} = \frac{1}{\sum_i \phi_i(s)} \sum_{i=1}^{N} \theta_a(i) \cdot \phi_i(s) \tag{3.13}$$

An dieser Stelle kann beobachtet werden, dass die approximierten Nutzenfunktionen nicht nur vom Parametervektor, sondern auch von der Wahl der Basisfunktionen sowie der Anzahl der Funktionen in der Basis für die Bestimmung des Basisvektors abhängig ist. Der Diskretisierungsgrad kann nach Feinheit mit der Anzahl N der verwendeten Basisfunktionen unterschieden werden. Je feiner die Zerlegung ist, desto besser ist die Approximation der tatsächlichen Nutzenfunktion, die erreicht werden kann. Je feiner die Zerlegung des Zustandsraumes ist, desto langsamer ist der Lernprozess. Der Lernprozess ist langsamer wegen einer großen Anzahl der benötigten Lernschritte, da mehrere Teile mehrere Male besucht werden müssen, um eine Konvergenz zur optimalen Bewertungsfunktion zu erreichen.

3.4.3 Tile-Coding und Coarse-Coding

Es gibt eine große Zahl an Möglichkeiten einen Zustandsraum zu diskretisieren, also die Basis der Funktionen eines Basisvektors zu wählen. Jede einzelne Basisfunktion aus dieser Basis soll einen individuellen Einzugsbereich (lokalisierende Eigenschaften) aufweisen.
Die erste, relativ populäre und einfache Diskretisierungsmethode ist die sogenannte Tile-Coding-Methode. Das Tile-Coding ist von Albus [Albus, 1971] unter dem Begriff ”cerebellar model articulator controller” eingeführt worden. Der Begriff Tile-Coding wurde von Watkins eingeführt, siehe [Watkins, 1989]. Die Basis besteht aus Impulsfunktionen $(\phi_1(), \dots \phi_N())$. Eine Impulsfunktion $\phi_i(s) = 1$ gibt den Wert 1 aus, wenn der Zustand im Einzugsbereich dieser Basisfunktion $\phi_i()$ liegt, sonst gibt sie den Wert 0 aus. Der Zustandsraum wird in gleich große Teile (engl.: tiles) aufgeteilt. Für jeden Teil wird der Einzugs-

[14] Wenn zum Beispiel als Basisfunktionen die Gauß’schen Funktionen gewählt werden, wird diese Gewichtung den Abstand des Merkmals zum Zustand repräsentieren.

bereich einer der Basisfunktionen aus dem Basisvektor definiert und hat einen Eintrag im Basisvektor $\mathbf{w}$. Der Basisvektor für den Zustand $s \in S$ hat an der Stelle i eine Eins, an der der Zustand im Einzugsbereich der Impulsfunktion $\phi_i(s)$ ist, sonst hat er nur Nullen: $\mathbf{w}(s) = (0, \ldots, 0, \underbrace{1}_{\text{Stelle } i}, 0, \ldots 0)^T$.
Ein Merkmalset ist ein Teil des Basisvektors, der nur die Nummern der Merkmale beinhaltet, in deren Einzugsbereich sich der aktuelle Zustand s befindet, das Merkmalset wir als $F(s)$ bezeichnet. Die gesamte Menge aller Teile wird als Tiling bezeichnet, so dass jeder Zustand aus dem kontinuierlichen Raum durch einen Teil des Tilings repräsentiert und durch eine Impulsfunktion im Basisvektor realisiert wird. Ein Tiling ist ein Gitter, das über den Zustandsraum gelegt wird, vergleiche Abbildung 3.1. Im Tile-Coding-Verfahren überlappen sich die Teile in einem Tiling nicht.
Alle Zustände in einem Teil der Tile-Coding-Methode sind für den Algorithmus von gleicher Bedeutung. Der Agent lernt für das gesamte Teil, wie gut bzw. schlecht die vorgenommene Aktion war. Gelernt wird mit dem Reinforcement-Learning-Ansatz, der für endliche Markow-Entscheidungsprobleme definiert ist. Eine bessere Approximation der Nutzenfunktion kann durch die Anwendung mehrerer Tilings für den Diskretisierungsschritt erreicht werden. Mehrere Tilings werden für die feinere Diskretisierung des Zustandsraums angewendet. Sei T die Anzahl der für die Diskretisierung bestimmten Tilings. Die Tilings liegen leicht verschoben übereinander. Der gesamte Basisvektor $\mathbf{w} := \mathbf{w_1} + \ldots \mathbf{w_T}$ wird aus einzelnen Basisvektoren $\mathbf{w_i}$ für das jeweilige Tiling $i = 1, \ldots, T$ gebildet. Der Basisvektor $\mathbf{w_i}$ des einzelnen Tilings i hat an der Stelle einen Eintrag 1, wo der aktuelle Zustand im Einzugsbereich des Teils des Tilings i liegt.
Die Q-Funktion für den aktuellen Zustand s und die Aktion a ist das Skalarprodukt aus dem Teil des Parametervektors θ_a für die Aktion a und dem gesamten Basisvektor $\mathbf{w}$. Jedes der Tilings ist für die Entscheidung über die auszuführende Aktion gleich verantwortlich.
Ein Beispiel für die Bestimmung der gesamten Basisvektoren für einen Zustand s, siehe mit einem Punkt gekennzeichneter Zustand in Abbildung 3.1, mit Anwendung zweier Tilings und Berechnung des $Q(s, \cdot)$-Wertes, wird an dieser Stelle beschrieben. Über den Zustandsraum werden zwei Gitter von zwei Tilings gelegt. Das erste Tiling wird in der Abbildung durch ein Gitter mit durchgezogenen Linien dargestellt. Das zweite Tiling wird mit punktierten Linien dargestellt. Die Nummerierung der Teile ist sequentiell $1, \ldots, 16$. Das Merkmalset besteht somit immer aus 2 Einträgen, jeder der Einträge ist die Nummer des Teils des jeweiligen Tilings. Der gesamte Gewichtsbasisvektor für den mit einem blauen Punkt gekennzeichneten Zustand s sieht wie folgt aus:
$\mathbf{w}(\mathbf{s}) = \frac{1}{2}(\underbrace{(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0)^T}_{\text{Tiling 1}} + \underbrace{(0,0,0,0,0,0,1,0,\ldots,0)^T}_{\text{Tiling 2}})$.
Das Merkmalset besteht aus zwei Einträgen $F(s) = \{3; 7\}$. Diese Einträge sind die Nummern der Teile, bei denen der Zustand im Einzugsbereich der Teile liegt.
Der Wert für die Q-Funktion wird aus dem Basisvektor und dem Parametervektor zusammengestellt, siehe auch Gleichung (3.13), so dass

$$Q(s,a) = {\theta_\mathbf{a}}^T \cdot \mathbf{w}(\mathbf{s}) = \frac{1}{2} \sum_{i \in \{3,7\}} \theta_a(i) \cdot \underbrace{\phi_i(s)}_{=1}.$$

Für die Coarse-Coding-Methode (CC-Methode) wird der Zustandsraum nicht wie im Tile-Coding einfach in gleiche Teile aufgeteilt, sondern es wird eine Basis aus lokalisierenden binären Funktionen aufgestellt. Der Basisvektor besteht, wie oben, aus den Einträgen der einzelnen lokalisierenden Funktion der gewählten Basisfunktionen. Die Basisfunktionen sind über den kontinuierlichen Zustandsraum verteilt und besitzen eine bestimmte Ausdehnung und Form. Die Basisfunktionen sind oft als sich gegenseitig

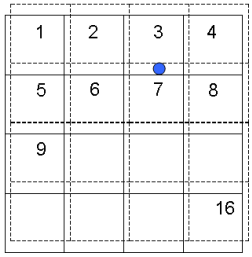


Abbildung 3.1: Diskretisierung des Zustandsraums mit der Tile-Coding-Methode. Zwei Tilings mit jeweils 16 Teilen.

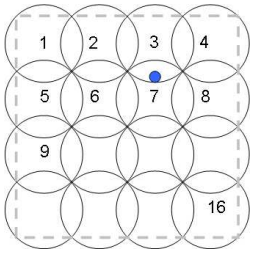


Abbildung 3.2: Diskretisierung des Zustandsraums mit der Coarse-Coding-Methode. 16 überlappende Teile.

überlappende Ellipsen, die über dem Zustandsraum liegen, definiert, siehe Abbildung 3.2. Die Basisfunktionen sind üblicherweise gegenseitig überlappend angeordnet, damit jeder mögliche Zustand durch mindestens ein Merkmal beschrieben wird. Die CC-Methode ist der oben beschriebenen Tile-Coding-Methode ähnlich, mit dem Unterschied, dass Überlappungen der einzelnen Teile erlaubt sind. Die unterschiedlichen Zustände können durch eine unterschiedliche Anzahl an Basisfunktionen repräsentiert sein. Die lineare Approximation der Q- und V-Funktionen unterscheidet sich im Vergleich zur Definition in [Sutton & Barto, 1998, Kap. 8.3, S. 201] im Normierungsfaktor, siehe Gleichungen (3.12), (3.13). Der Grund dafür liegt in der Beobachtung, dass die nicht normierte Q-Funktion mit einer Repräsentation der Merkmale durch die CC-Methode zu einer Unausgewogenheit führt. Zum Beispiel liefern zwei Zustände, die nah aneinander liegen, aber durch eine unterschiedliche Anzahl an Merkmalen, in deren Einzugsbereich der Zustand liegt, repräsentiert sind, unterschiedliche Werte für die Q-Funktion. Wenn ein Zustand durch Ausführen der Aktion vom nahe liegenden Zustand erreicht wird, dann enthält der gebildete TD-Fehler unterschiedlich dimensionierte Q-Werte, und entspricht nicht dem tatsächlichen Fehler, sondern hängt zusätzlich von der Anzahl der die Q-Funktion repräsentierenden Merkmale ab.
Beispiel: Zwei Zustände $s_1 \approx s_2$ liegen nah beieinander. Ein Zustand wird durch zwei Merkmale repräsentiert, der andere Zustand wird durch ein Merkmal repräsentiert. Die Q-Funktionen werden dann wie folgt gebildet:

$$Q(s_1, a) = \sum_{i \in \{3,7\}} \theta_a(i) = \theta_a(3) + \theta_a(7)$$

$$Q(s_2, a) = \sum_{i \in \{3\}} \theta_a(i) = \theta_a(3)$$

Es wird angenommen, dass die Werte im Parametervektor für die nahe liegenden Zustände ähnlich sind. In diesem Fall werden sich die Q-Werte trotz sehr ähnlicher Situationen für die nah beieinander liegenden Zustände dramatisch unterscheiden. $Q(s_2, a) = \frac{1}{2} Q(s_1, a)$.
Wenn die Anzahl der Einträge im Merkmalset für jeden Zustand $s \in S$ immer gleich bleibt, wird in [Sutton & Barto, 1998] vorgeschlagen, den angesprochenen Normierungsschritt im Lernparameter $\frac{\alpha}{|F(s)|}$ einzubauen.

3.4.4 Codierung mit RBF

Die Verwendung von radialen Basisfunktionen (RBF) für die Zerlegung des Zustandsraumes ist eine Verallgemeinerung der CC-Methode, in der die Basisfunktionen als einfache Impulsfunktionen definiert sind. Wenn aber an Stelle der Impulsfunktionen die RBF angewendet werden, wird von RBF-Netzwerken

gesprochen. Eine RBF nimmt Werte aus dem ganzen Intervall $[0, 1]$ [15] und nicht nur binäre Ausprägungen an. Jede der RBF misst die Nähe des aktuellen Zustands zu ihrem Zentrum.
Der Zustandsraum wird von überlappenden RBF abgedeckt. Die Ausprägungen der einzelnen Funktionen in Bezug auf den aktuellen Zustand werden in einem Basisvektor zusammengefasst.
Ein RBF-Merkmal $\phi_j()$ aus dem Basisvektor $(\phi_1(), \ldots, \phi_N())^T$ wird als eine Gauß'sche Glockenkurve definiert, die durch die Ausdehnung des Merkmals $\mathbf{\Sigma_j^{-1}}$ und die Position $\mu_\mathbf{j}$ des Merkmals bestimmt wird, siehe Gleichung (3.14).

$$\phi_j(s) = \exp(-\frac{1}{2} \cdot (\mathbf{s} - \mu_\mathbf{j})^T \Sigma_j^{-1} \cdot (\mathbf{s} - \mu_\mathbf{j})) \tag{3.14}$$

Es wird angenommen, dass die Komponenten des Zustandes voneinander unabhängig sind. Also ist für den Zustandsraum mit der Dimension K die Kovarianzmatrix wie folgt definiert:

$$\Sigma_j^{-1} = \begin{pmatrix} \frac{1}{\sigma_j(1)^2} & 0 & 0 & \ldots & 0 \\ 0 & \frac{1}{\sigma_j(2)^2} & 0 & \ldots & 0 \\ 0 & 0 & \frac{1}{\sigma_j(3)^2} & \ldots & 0 \\ & & \ldots & & \\ 0 & 0 & 0 & \ldots & \frac{1}{\sigma_j(K)^2} \end{pmatrix} \tag{3.15}$$

$\sigma_\mathbf{j} := (\sigma_j(1), \ldots, \sigma_j(K))^T$ wird als der Vektor aller Ausdehnungen der RBF bezeichnet. Die Kovarianzmatrix ist mit dem Vektor $\sigma_\mathbf{j}$ wie folgt $\Sigma_j := I \cdot \sigma_\mathbf{j}$ verbunden, wobei I eine Einheitsmatrix bezeichnet. In diesem Fall kann die Basisfunktion wie folgt umgestellt werden:

$$\phi_j(s) = \exp^{-\frac{1}{2}\sum_{k=1}^{K} \frac{(s(k)-\mu_j(k))^2}{\sigma_j(k)^2}}.$$

Für den Zustand s besteht das Merkmalset $F(s)$ aus den Merkmalen, in deren Einzugsbereich sich der aktuelle Zustand befindet. Der Zustand s befindet sich im Einzugsbereich eines Merkmals $\phi_i(s)$, wenn der berechnete Wert $\phi_i(s)$ größer als ein vorgegebener Schwellenwert $Sensitivität$ ist. Dieser Wert wird in der vorliegenden Arbeit auf 0.3 gesetzt.

$$F(s) := \{j : \phi_j(s) > Sensitivität\} \tag{3.16}$$

$$Q(s,a) = \sum_{j \in F(s)} \theta_a(j) \cdot \underbrace{\frac{\phi_j(s)}{\sum_{k \in F(s)} \phi_k(s)}}_{:= w_j(s)} = \sum_{j \in F(s)} \theta_a(j) \cdot w_j(s) = \frac{1}{\sum_{k \in F(s)} \phi_k(s)} \cdot {\theta_\mathbf{a}}^T \cdot \phi(\mathbf{s}) \tag{3.17}$$

Wenn der Basisvektor als Vektor der RBF repräsentiert wird, wird die Q-Funktion durch ein RBF-Netzwerk approximiert. Die Bestimmung des Merkmalsets $F(s)$ für den aktuellen Zustand s für die Diskretisierung mit Codierung der RBF kann wie folgt bestimmt werden. Zuerst wird die am nächsten zum aktuellen Zustand s liegende RBF bestimmt, deren Index als $Index_s$ bezeichnet wird. Dieses Zentrum wird zum leeren Merkmalset $F(s)$, das für den aktuellen Zustand bestimmt werden soll, hinzugefügt. Von diesem Zentrum aus werden dann die Nachbarn bestimmt, die den Zustand s noch im Einzugsbereich beinhalten. Dafür wird der Parameter $Anzahl$ vorgegeben, der in Abhängigkeit vom Parameter $Sensibilität$ definiert ist. Für diese Zentren $i = -Anzahl, \ldots, Anzahl$ wird geprüft, ob der Wert der Basisfunktion $\phi_{Index_s+i}(s)$ für den aktuellen Zustand s den vorgegebenen Schwellenwert

[15] wenn als RBF die Gauß-Funktionen gewählt werden

$Sensibilität$ überschreitet. Wenn dieser Fall auftritt, wird dieser Index $Index_s + i$ zum Merkmalset $F(s)$ hinzugefügt.
In Abbildung 3.3 ist das RBF-Netzwerk, das die Q-Funktionen approximiert, abgebildet. In der verdeckten Schicht der RBF-Netzwerke liegen Basisfunktionen, deren Vereinigung den Zustandsraum beinhaltet. Die Anzahl der Knoten in der verdeckten Schicht der RBF-Netzwerke ist gleich der Anzahl der Merkmale im Basisvektor.
Es gibt unterschiedliche Vorgehensweisen solche Netzwerke zu erlernen. Die erste Möglichkeit besteht darin, nur die Gewichte der Ausgabeschicht des RBF-Netzes, also des Parametervektors, zu lernen. Die andere Möglichkeit besteht darin, die zusätzlichen Parameter der Basisfunktionen auch während des Lernprozesses zu adaptieren.
Bei der ersten Möglichkeit können die Knoten in der verdeckten Schicht gleichmäßig über den Zustands-

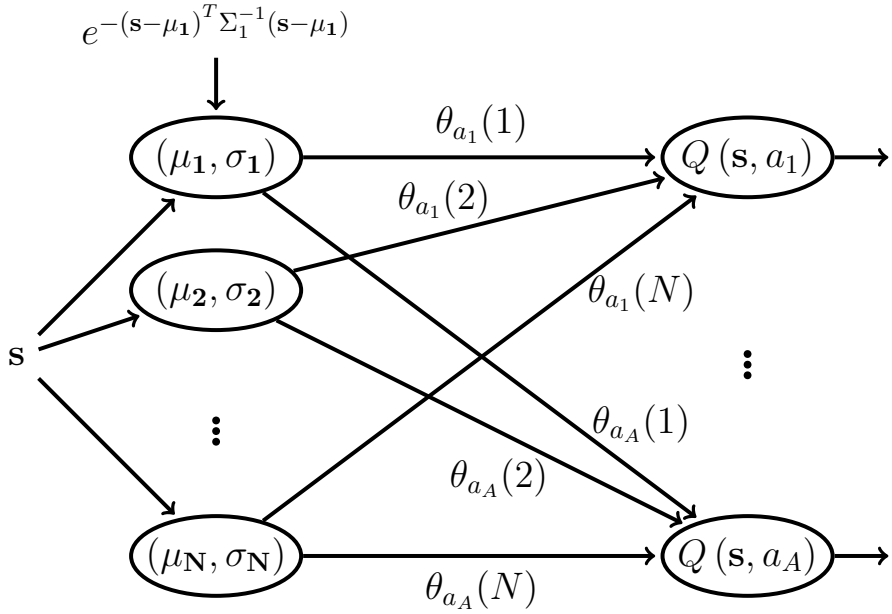


Abbildung 3.3: Codierung der Q-Funktion mit einem RBF-Netzwerk

raum verteilt werden. Die Parameter der RBF werden als konstante Werte vorgegeben. In diesem Fall unterscheidet sich das Verfahren nicht vom linearen Gradientenabstiegs-SARSA(λ)-Algorithmus, siehe [Sutton & Barto, 1998, Kap. 8.4, S. 212]. Es wird nur der Parametervektor gelernt, der im linearen Zusammenhang mit dem Basisvektor steht.
Die zweite Variante besteht in der Adaption des Parametervektors der Ausgabeschicht θ und der Parameter der verdeckten Schicht (μ, σ). Durch Bildung des Skalarprodukts aus dem Parametervektor θ und dem Basisvektor $(\phi_1() \sim N(\mu_\mathbf{1}, \sigma_\mathbf{1}), \ldots, \phi_N() \sim N(\mu_\mathbf{N}, \sigma_\mathbf{N}))$ wird der Q-Wert gebildet. Die zusätzlichen Lernschritte und der zusammengestellte Algorithmus 3.2 sind im nächsten Kapitel beschrieben. Der vorgeschlagene Adaptionsschritt ist für den Lernprozess im Onlinebetrieb geeignet.

3.4.5 RBF-Netzwerk als nichtlineare Approximation der Q-Funktion

In diesem Abschnitt wird eine weitere Möglichkeit eingeführt die Nutzenfunktion durch das ”echte” RBF-Netzwerk zu approximieren. In einem ”echten” RBF-Netzwerk besteht der Parametervektor nicht nur aus dem Gewichtsbasisvektor zwischen der verdeckten und der Ausgabeschicht, sondern auch aus

Parametern in der verdeckten Schicht des RBF-Netzwerks, vergleiche Abbildung 3.3. Die Möglichkeit der Approximation der Nutzenfunktion mit einem RBF-Netzwerk wird schon in [Poggio & Girosi, 1989] beschrieben.
Die Parameter in der verdeckten Schicht sollen während des Lernprozesses auch adaptiert werden. Ein zusätzlicher Adaptionsschritt für die Parameter in der verdeckten Schicht wird in diesem Abschnitt eingeführt. Durch den zusätzlichen Adaptionsschritt wird die Position und Ausdehnung der Basisfunktionen feiner justiert, dadurch entsteht eine große Flexibilität der Approximationsfunktion für die gesuchte Nutzenfunktion.
Der Unterschied zum gewöhnlichen RBF-Netzwerk liegt in der gebildeten Fehlerfunktion, die nicht aus dem quadratischen Fehler zwischen der Antwort auf den vorgegebenen Input, der durch die Trainingsmenge vorgegeben wird, und der Antwort, die durch Propagierung der Inputwerte durch das RBF-Netzwerk erzielt wird, berechnet wird, wie es im überwachten Lernen der Fall ist, sondern aus dem TD-Fehler abgeleitet wird. Aus diesem Grund kann der Ansatz der linearen Regression nicht angewendet werden. Der Schritt der Parameteradaption in beiden Schichten des RBF-Netzwerks wird durch stochastische Gradientenabstiegsverfahren realisiert.
In den Arbeiten von Menache [Menache et al., 2005] und [Menache, 2003] wird eine ähnliche Idee wie in diesem Abschnitt verfolgt. Reinforcement-Learning wird mit dem Lernen mit RBF-Netzen verbunden. Die Zentren der für die Approximation der Bewertungsfunktion verwendeten RBF-Netze und deren Ausdehnung werden mitgelernt, wie der Parametervektor der Ausgabeschicht. Die von Menache vorgeschlagene Methode basiert auf der Methode des Least Squares Temporal Difference Learning (LSTD). LSTD ist eine Erweiterung des SARSA-Algorithmus, die schon in [Bradtke et al., 1996] vorgeschlagen ist. Zum Beispiel sind die Algorithmen der LSTD in [Boyan, 2002] mit der Erläuterung der grundlegenden Idee eingeführt.
Im Rahmen der vorliegenden Arbeit werden zusätzliche Adaptionsschritte der Parameter der verdeckten Schicht als Erweiterung des SARSA-Algorithmus eingeführt.
Für den Zustand s_t besteht das Merkmalset $F(s_t)$ aus Merkmalen, in deren Einzugsbereich sich der Zustand s_t befindet, $F(s_t) := \{j : \phi_j(s_t) > Sensitivität\}$. Die Aktion a_t wird durch eine ϵ-gierige Strategie $a_t := \pi(s_t) = \epsilon - gierigQ(s_t, \cdot)$ bestimmt. N ist die Anzahl der Basisfunktionen im Basisvektor, $Norm = \sum_{j \in F(s_t)} \phi_j(s_t)$ ist der Normierungsfaktor zur Bildung des Gewichtsbasisvektors, $w_j(s_t) := \frac{1}{Norm} \cdot \phi_j(s_t)$ ist ein Gewicht des Parameterwerts $\theta_{a_t}(j)$, $j \in F(s_t)$.
Der Adaptionsschritt für die Parameter der Basisfunktionen im RBF-Netzwerk wird durch Gleichung (3.18) für die Zentren und Gleichung (3.19) für die Ausdehnung repräsentiert. Dieser Adaptionsschritt für Parameter des RBF-Netzwerks in der verdeckten Schicht ist für den Onlinebetrieb konzipiert.
Der Temporal-Difference-Fehler $\delta_t^2 = \frac{1}{2}(r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))^2$ beinhaltet die Abhängigkeit von den Zentren der Basisfunktionen und kann durch Anwendung der stochastischen Gradientenabstiegsverfahren (Bildung der Gradienten in μ_j, $j \in F(s_t)$) für die Fehlerreduktion dienen. Für jedes $j \in F(s_t)$ gilt

$$\mu_{\mathbf{j}}^{\mathbf{neu}} = \mu_{\mathbf{j}}^{\mathbf{alt}} + \alpha_\mu \cdot \delta_t \cdot \theta_{a_t}(j) \cdot w_j(s_t) \cdot (w_j(s_t) - 1) \cdot \Sigma_j^{-1} \cdot (\mathbf{s_t} - \mu_{\mathbf{j}}^{\mathbf{alt}}) \tag{3.18}$$

Die Kovarianzmatrix Σ_j ist in Gleichung (3.15) definiert.
Die Herleitung des Lernschrittes (3.18) wird in Anhang A durchgeführt. Die Ausdehnungen $\sigma_{\mathbf{j}}$ der Gaußkurve für jedes $j \in F(s_t)$, vergleiche Gleichung (3.15), werden so umgerechnet, dass jeder Zustand $s_t \in S$ aus dem Zustandsraum durch mindestens eines der Merkmale (radiale Basisfunktion) repräsentiert ist. K ist die Dimension des Zustandsraumes. Die Ausdehnung der Gauß'schen Funktion $\sigma_j(k)$ wird für jedes Element aus dem Merkmalset $j \in F(s_t)$ und jede Koordinate $k \in 1, \ldots, K$ des Vektors

$(\sigma_j(1), \ldots, \sigma_j(K))^T$ wie folgt umgerechnet:

$$\sigma_j^{neu}(k) = Parameter_\sigma \cdot \max(|\mu_j(k) - \mu_{j_{links}}(k)|, |\mu_j(k) - \mu_{j_{rechts}}(k)|) \tag{3.19}$$

wobei mit $\mu_{j_{links}}(k)$ der linke Nachbar und mit $\mu_{j_{rechts}}(k)$ der rechte Nachbar des j-ten Elements des Basisvektors in der Koordinate k bezeichnet wird. Zum Beispiel sind μ_3 und μ_5 Nachbarn für das Element μ_4 in der Koordinate x_1, siehe Abbildung 3.4. Mit dem Parameter $Parameter_\sigma$ kann die Ausdehnung der Gauß'schen Basisfunktion gesteuert werden.
Zur Verdeutlichung des Schrittes der Gleichung (3.19) wird an dieser Stelle ein Beispiel erläutert. Sei der Zustandsraum ein zweidimensionaler Raum (x_1, x_2). Sei $Tiles_{x_1}$ die Anzahl der Teile für die Zerlegung in x_1-Richtung und $Tiles_{x_2}$ die Anzahl der Teile in x_2-Richtung, siehe Abbildung 3.4. Jeder der Merkmale für den Basisvektor sei wie folgt angeordnet: für $j \in \{0, \ldots, Tiles_{x_1} * Tiles_{x_2} - 1\}$ gilt $j := l_j * Tiles_{x_1} + k_j$, ($l_j \in \{0, \ldots, Tiles_{x_2} - 1\}$, $k_j \in \{0, \ldots, Tiles_{x_1} - 1\}$). Insgesamt sind $Tiles_{x_1} * Tiles_{x_2}$ Einträge im Basisvektor enthalten. Für den $\sigma_\mathbf{j}$-Parameter, $j \in F(s)$, wird Gleichung (3.19) wie folgt angewendet.

$$\sigma_j(1) = \max(|\mu_j(1) - \mu_{(l_j \cdot Tiles_{x_1} + k_j - 1)}(1)|, |\mu_j(1) - \mu_{(l_j \cdot Tiles_{x_1} + k_j + 1)}(1)|)$$

$$\sigma_j(2) = \max(|\mu_j(2) - \mu_{(l_j - 1) \cdot Tiles_{x_2} + k_j}(2)|, |\mu_j(2) - \mu_{(l_j + 1) \cdot Tiles_{x_2} + k_j}(2)|)$$

Das beschriebene Beispiel wird an dieser Stelle graphisch verdeutlicht. Der Zustandsraum ist ein 2D-

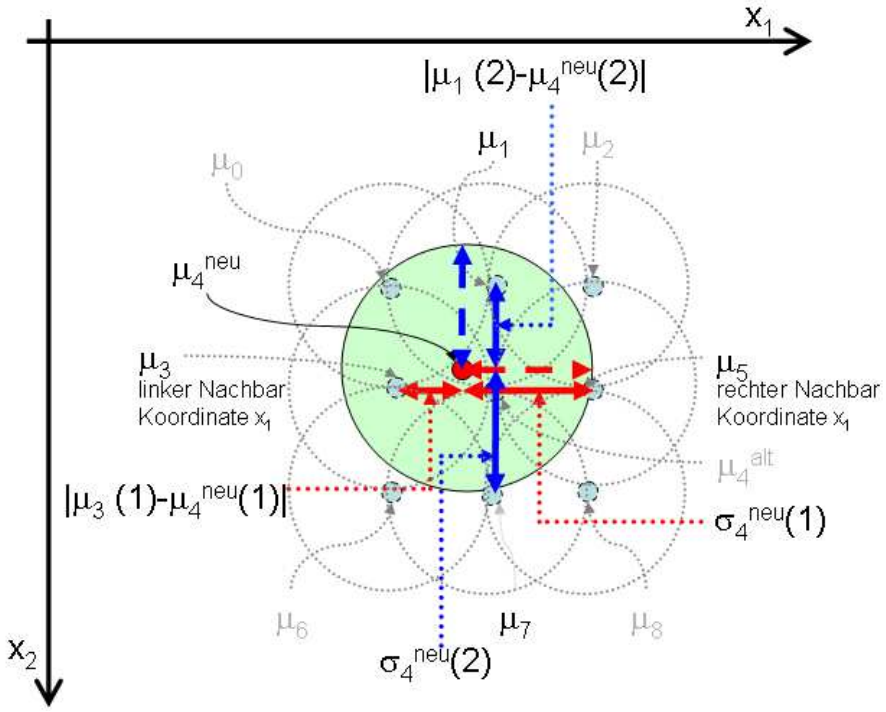


Abbildung 3.4: Grafische Darstellung der Anpassung des $\sigma_\mathbf{4}$-Parameters für ein neu bestimmtes Zentrum $\mu_\mathbf{4}^\mathbf{neu}$, siehe Gleichung (3.19)

Raum. Die Anzahl der Merkmale im Basisvektor ist $9 := Tiles_{x_1} \cdot Tiles_{x_2}$, wobei $Tiles_{x_1} = 3$, $Tiles_{x_2} = 3$. Die Merkmale sind gleichmäßig über den Zustandsraum verteilt, siehe Abbildung 3.4. Nach Anwendung der Gleichung (3.18) ist das Zentrum des Merkmals Nummer $4 := 1 \cdot 3 + 1$ verschoben,

$\mu_{\mathbf{4}} := (\mu_4(1), \mu_4(2))^T$. Das neue Zentrum $\mu_{\mathbf{4}}^{\mathbf{neu}}$ ist mit einem roten Punkt gekennzeichnet. Dann wird der Parameter $\sigma_{\mathbf{4}}$ so verändert, dass die Überlappungen noch gewährleistet werden können, also $\sigma_4(1) = \max(|\mu_4^{neu}(1) - \mu_3(1)|, |\mu_4^{neu}(1) - \mu_5(1)|)$, wobei $\mu \underbrace{3}_{:=1\cdot 3+1-1}(1)$ der linke Nachbar in der Koordinatenachse x_1 ist, $\mu \underbrace{5}_{:=1\cdot 3+1+1}(1)$ ist der rechte Nachbar. Diese beiden Werte sind mit roten Pfeilen gekennzeichnet. Und für die x_2-Achse gilt dasselbe Prinzip $\sigma_4(2) = \max(|\mu_4^{neu}(2) - \mu_1(2)|, |\mu_4^{neu}(2) - \mu_7(2)|)$, diese Werte sind in Abbildung 3.4 mit einem blauen Pfeil gekennzeichnet. Die neuen Werte für die Ausdehnung der RBF sind mit gestrichelten Pfeilen neben $\mu_{\mathbf{4}}^{\mathbf{neu}}$ in der Abbildung gekennzeichnet.

3.4.6 Gradientenabstiegs-SARSA-Algorithmus

Der im vorliegenden Abschnitt eingeführte Gradientenabstiegs-SARSA-Algorithmus ist der SARSA-Algorithmus für einen kontinuierlichen Zustandsraum. Der Gradientenabstiegs-SARSA-Algorithmus für binäre Basisfunktionen im Basisvektor und eine ϵ-gierige Strategie aus [Sutton & Barto, 1998, Kap.8.4, S. 212] wird als Grundlage für den hier eingeführten Algorithmus übernommen. Der vorliegende Algorithmus ist auch für eine Basis anwendbar, die nicht nur aus binären Basisfunktionen besteht. Die Basis wird am Anfang des Algorithmus initialisiert, d.h. die Diskretisierungsmethode muss am Anfang ausgewählt werden. Wenn die Nutzenfunktion durch ein "echtes" RBF-Netzwerk approximiert wird, werden zwei zusätzliche Abschnitte eingefügt, die besonders kennzeichnet werden. Diese Abschnitte beinhalten die hergeleiteten Formeln (3.18) und (3.19), mit deren Hilfe die Parameter der verdeckten Schicht des "echten" RBF-Netzwerks, also die Parameter der Basisfunktionen, im Onlinebetrieb zusammen mit den Parametern der Ausgabeschicht adaptiert werden können. Bei der Adaption der Zentren der Basisfunktionen wird eine Einschränkung eingeführt. Die Zentren dürfen nicht weiter wandern als bis zu den in der Nachbarschaft liegenden, gleichmäßig verteilten Zentren, wie es am Anfang mit $\mu^{Orig.}$ vorinitialisiert wurde, siehe Gleichung (3.20).
Der Gewichtsbasisvektor wird als $\mathbf{w}(\mathbf{s_t})$ bezeichnet. Die Verbindung zwischen dem Gewichtsvektor und den RBF-Zentren μ ist in den Gleichungen (3.14) und (3.11) aufgestellt. Das Merkmalset $F(s_t)$ beinhaltet die Indizes der Basisfunktionen, in deren Einzugsbereich der aktuelle Zustand s_t liegt.

$$\mu_j(k) \leftarrow \begin{cases} \mu_j(k) + \delta_\mu(k) & \text{wenn} \quad \mu_j^{Orig.}(k-1) < \mu_j(k) + \delta_\mu(k) < \mu_j^{Orig.}(k+1) \\ \mu_j^{Orig.}(k-1) & \text{wenn} \quad \mu_j(k) + \delta_\mu(k) < \mu_j^{Orig.}(k-1) \\ \mu_j^{Orig.}(k+1) & \text{wenn} \quad \mu_j(k) + \delta_\mu(k) > \mu_j^{Orig.}(k+1) \end{cases} \tag{3.20}$$

Der Grund für diese Einschränkung liegt darin, dass das Lernen am Anfang anhand von ungenauen Messungen der Q-Werte abläuft. Dies führt zu einer relativ schnellen Verstimmung der Abdeckung des Zustandsraums durch RBF und als Folge zur Verschlechterung des Lernprozesses, so dass viele Lernschritte benötigt werden, um diesen eingebauten Fehler zu beseitigen, siehe Zeile 35 des Algorithmus.
Die Adaption der RBF-Zentren wird gemäß Gleichung (3.20) realisiert, siehe Zeile 5 des Algorithmus. Die Adaption der Ausdehnung der RBF (σ) wird wie im angesprochenen Beispiel bzw. in der aufgestellten Gleichung (3.19) durchgeführt.
Ein wichtiger Parameter, wenn die verdeckte Schicht des RBF-Netzwerks mitgelernt wird, ist der Lernschrittparameter α_μ, siehe Zeile 33 des Algorithmus.
Die Auswirkung des Lernparameters α_μ auf den Lernprozess in der Approximation der Q-Funktion durch ein "echtes" RBF-Netzwerk wird in Kapitel 3.7.2 untersucht. Die Ergebnisse gelten für das Mountain-Car-Problem. Dabei wird Lernen mit und ohne Adaption der Parameter der RBF in der verdeckten

Algorithmus 3.2 Gradientenabstiegs-SARSA-Algorithmus nach [Sutton & Barto, 1998, Figure 8.8, S. 212]

Initialisiere θ-Parameter
Initialisiere den Basisvektor $(\phi_1(), \ldots \phi_N())^T$ ▷ Gleichverteilt und überlappend
*** ▷ Anfang: Ausschnitt nur für "echtes" RBF-Netzwerk
μ, Σ^{-1}-Vektoren ▷ Für RBF-Netzwerk
$\mu^{Orig.} := \mu$
*** ▷ Ende: Ausschnitt nur für "echtes" RBF-Netzwerk
repeat Für jede Episode $episode$ ▷ $episode = 0, \ldots, E$
    $t \leftarrow 0$
    Initialisiere s_0
    for $a \in A(s_0)$ **do**
        $\mathbf{e_a} \leftarrow \mathbf{0}$ ▷ Setze alle Spuren der Eligibility-Traces auf Null
    $F(s_0)$ ▷ Bestimme das s_0 repräsentierende Merkmalset
    $\mathbf{w(s_0)} = \frac{1}{\sum_{i \in F(s_0)} \phi_i(s_0)} (\phi_1(s_0), \ldots \phi_N(s_0))^T$
    for $a \in A(s_0)$ **do**
        $Q(s_0, a) \leftarrow \sum_{i \in F(s_0)} \theta_a(i) \cdot w_{s_0}(i)$
    $a_t \leftarrow \epsilon - gierig(Q(s_t, \cdot))$ ▷ Neue Aktion mit ϵ-gieriger Strategie
    repeat Für jeden Schritt t der $episode$ ▷ $t = 0, \ldots, T$
        for $a \in A(s_t)$ **do**
            $\mathbf{e_a} \leftarrow \alpha\lambda\mathbf{e_a}$ ▷ Aktualisiere Eligibility-Traces
        for alle $i \in F(s_t)$ **do**
            $e_{a_t}(i) \leftarrow w_{s_t}(i)$
        Führe die Aktion a_t aus, beobachte r_t, s_{t+1}
        $\delta_t = r_t - Q(s_t, a_t)$
        $F(s_{t+1})$ ▷ Bestimme Merkmalset in s_{t+1}
        $\mathbf{w(s_{t+1})} = \frac{1}{\sum_{i \in F(s_{t+1})} \phi_i(s_{t+1})} (\phi_1(s_{t+1}), \ldots \phi_N(s_{t+1}))^T$
        for $a \in A(s_{t+1})$ **do**
            $Q(s_{t+1}, a) \leftarrow \sum_{i \in F(s_{t+1})} \theta_a[i] \cdot e[i]$
        $a_{t+1} \leftarrow \epsilon - gierig(Q(s_{t+1}, \cdot))$ ▷ Neue Aktion mit ϵ-gieriger Strategie
        $\delta_t \leftarrow \delta_t + \gamma Q(s_{t+1}, a_{t+1})$
        $\theta_{\mathbf{a_t}} \leftarrow \theta_{\mathbf{a_t}} + \alpha\delta_t\mathbf{e_{a_t}}$
        *** ▷ Anfang: Ausschnitt nur für "echtes" RBF-Netzwerk
        for $j \in F(s_t)$ **do**
            $\delta_\mu = \alpha_\mu \cdot \delta_t \cdot \theta_{a_t}(j) \cdot e_{a_t}(j)(e_{a_t}(j) - 1) \cdot \Sigma_j^{-1} \cdot (\mathbf{s_t} - \mu_\mathbf{j})$
            for alle $k = 0, \ldots, |s_t|$ **do**
                $\mu_j(k)$ neu berechnen gemäß Gleichung (3.20)
            σ_j neu berechnen gemäß Gleichung (3.19)
        *** ▷ Ende: Ausschnitt nur für "echtes" RBF-Netzwerk
        $t \leftarrow t + 1$
    until s_{t+1} der Zielzustand ist
until $episode < E$

Schicht verglichen. Es wird experimentell gezeigt, dass der Lernparameter α_μ eine sehr große Wirkung auf die Verbesserung des Lernprozesses ausübt.

3.5 Modifizierte Lernregel für das Semi-Markow-Entscheidungsproblem

Wenn die Aktionen eine unterschiedliche Zeitdauer zur Ausführung benötigen, wird der Prozess nicht als Markow-Entscheidungsproblem, sondern als Semi-Markow-Entscheidungsproblem (SMDP) bezeichnet. Ein SMDP ist eine Verallgemeinerung eines Markow-Entscheidungsproblems. Im Unterschied zu einem Markow-Entscheidungsproblem, dessen Zustandsänderungen in gleichen Zeitabständen erfolgen, wird hierbei die Verweildauer in einem Zustand durch einen weiteren stochastischen Prozess vorgegeben, [Wikipedia, 2011d].
Ein SMDP kann durch ein 5-Tupel beschrieben werden $(S, A, P^a_{ss'}, R^a_{ss'}, F^a_{st})$, wobei die ersten 4 Komponenten wie bei Markow-Entscheidungsproblemen definiert sind, siehe Kapitel 3.2.1. Die Komponente F^a_{st} gibt die Wahrscheinlichkeit dafür an, dass die Ausführung der Aktion a im Zustand s nach t Zeitschritten beendet ist. Die modifizierte Lernregel für die Q-Funktion ist im Semi-Markow-Entscheidungsproblem nach [Parr, 1998, Kap. 3.2, S. 38] im Zustand s, von dem aus der Zustand s' durch die Ausführung der Aktion a in t Zeitschritten und für die Belohnung r erreicht wird, wie folgt formuliert:

$$Q^{neu}(s,a) \leftarrow Q^{alt}(s,a) + \alpha(s,a) \cdot (r + \gamma^t \cdot \max_{a'} Q^{alt}(s',a') - Q^{alt}(s,a))$$

Die Lernrate $\alpha(s,a)$ hängt von dem Zustand s, der Aktion a und der Anzahl der durchgeführten Lernschritte ab. Der Unterschied zum SARSA-Lernschritt liegt im hinzugefügten Faktor γ^t, durch den die in Anspruch genommenen Zeiten für die Ausführung unterschiedlicher Aktionen simuliert werden.
Der SMDP wird im DPA-RL-Agenten zur Geltung kommen. Die vom DPA-RL-Agenten ausgeführten abstrakten Aktionen haben eine unterschiedliche Zeitdauer.

3.6 Messkriterien und Signifikanz der Ergebnisse

Für die Bestimmung der Qualität der entwickelten Agenten bzw. für die Untersuchung der unterschiedlichen Konfigurationen von Parametersätzen werden zwei Messkriterien in diesem Unterkapitel aufgestellt. Mithilfe dieser Messkriterien, unspezifisch als *Mess* bezeichnet, werden die in der vorliegenden Arbeit erzielten Ergebnisse miteinander verglichen. Um die Aussagekraft der Vergleiche zu erhöhen wird auch die Signifikanz der erzielten Unterschiede in den Ergebnissen untersucht. Für die Ermittlung der signifikanten Aussagen bezüglich der ermittelten Ergebnisse wird eine Statistik aufgestellt, die aus Ergebnissen mehrerer Versuche unter gleichen Bedingungen besteht. Unterschiede zwischen Ergebnissen aus Versuchen unter gleichen Bedingungen resultieren unter anderem aus der Anwendung der Zufallsvariable ϵ und dem Diskretisierungsschritt des Zustandsraumes. Eine Stichprobe dieser Statistik wird für eine untersuchte Konfiguration der angewendeten Lernmethode erstellt. Der Stichprobenumfang enthält N Versuche[16].
Die Signifikanzprüfung wird nach [Hartung, 1985, S. 510f] mithilfe des t-Tests vorgenommen. Zwei Stichproben zweier Konfigurationen sollen verglichen werden. Seien $\bar{Mess}_1$, $\bar{Mess}_2$ die Mittelwerte der untersuchten Messkriterien beider Stichproben und v_1, v_2 die erwartungstreuen Standardabweichungen beider Stichproben. Es sei $c := t(1 - \frac{\beta}{2}, 2 \cdot N - 2)$ das Quantil der t-Verteilung für $2N - 2$ Freiheitsgrade

[16] Je nach Experiment beträgt der Stichprobenumfang 30, 50 oder 200 Versuche.

auf dem Signifikanzniveau $\beta = 5\%$ für einen Test von zwei unabhängigen Stichproben. Das Quantil der t-Verteilung wird aus der t-Tabelle ausgelesen. Beispielsweise ist für einen Stichprobenumfang von 50 das Quantil der t-Verteilung $c = t(0.975, 98) = 1.98$. Für den Stichprobenumfang 30 hat das Quantil der t-Verteilung den Wert $c = 2.042$.
Dann liegt ein signifikanter Unterschied im Messkriterium $Mess_i$ zweier Stichproben $i = 1, 2$ vor, wenn die vorliegende Ungleichung (3.21) gilt. v_i^2 steht für die Standardabweichung im Messkriterium $Mess_i$ in der Stichprobe i.

$$|\bar{Mess}_1 - \bar{Mess}_2| > \frac{c}{\sqrt{N}} \cdot \sqrt{|v_1^2 + v_2^2|} \tag{3.21}$$

Zwei Messkriterien, auf deren Grundlage die Aussage über die Lernqualität der untersuchten Konfiguration getroffen wird, liegen vor. Diese Messkriterien sind der Lernfortschritt, bezeichnet als LP und die Stoßzahl, bezeichnet als SZ, also $Mess \in \{LP, SZ\}$. Es werden ein paar Hilfskriterien zur Messung der untersuchten Lernqualität eingeführt.
Die *Lernkurve* repräsentiert die Abhängigkeit der bis zum Ziel benötigten Anzahl an Schritten von der entsprechenden Episode. Dabei enthält jeder Punkt einen Mittelwert über N durchgeführte Versuche. Der Verlauf der Kurve kann einen Überblick über den Lernverlauf verschaffen. Je schneller die Kurve zu einem Wert konvergiert und je kleiner dieser Wert ist, desto schneller und erfolgreicher ist der Lernvorgang verlaufen.
Der *Lernfortschritt* (engl.: learning progress, $Mess = LP$) wird als über N Elemente der Stichprobe gemittelte durchschnittliche Anzahl der bis zum Ziel benötigten Schritte über den ersten E Episoden definiert. Dies entspricht der Fläche unterhalb der gemittelten Lernkurve im Intervall $[0, E]$, siehe Abbildung 3.5. Je niedriger dieser Wert ist, desto schneller wird der Lernprozess verlaufen, desto besser ist die zu testende Konfiguration.
Die Varianz (Standardabweichung) des Lernfortschritts v^{LP} gibt an, wie stabil die betrachtete Stichpro-

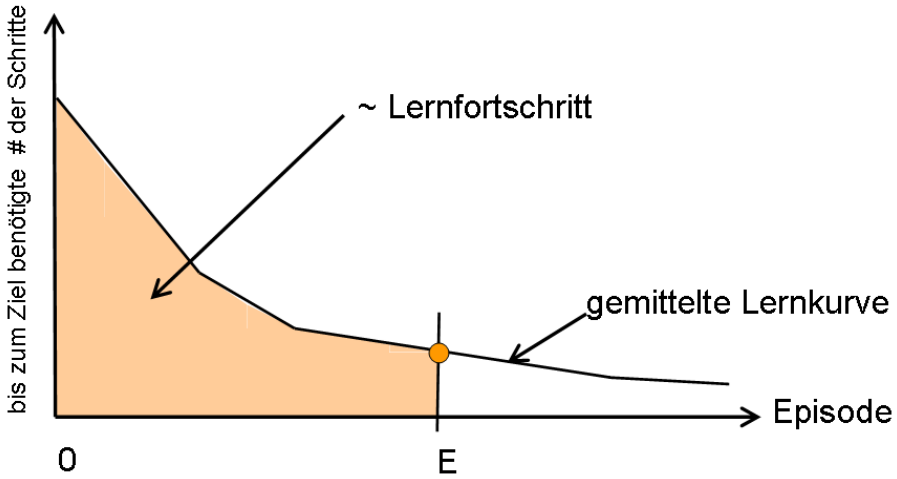


Abbildung 3.5: Das Messkriterium Lernfortschritt entspricht der Fläche unter der Lernkurve für die Episoden 0 bis E

be ist. Dieses Hilfskriterium berechnet die Abweichung des für die Stichprobe berechneten mittleren Lernfortschritts vom jeweiligen Lernfortschritt, der für die einzelnen Elemente der Stichprobe berechnet wird. Die Varianz wird zur Prüfung der Aussagekraft der Ergebnisse mithilfe des oben beschriebenen t-Tests verwendet.

Für die untersuchte Navigationsaufgabe ist nicht nur die durch den Lernfortschritt repräsentierte Lerngeschwindigkeit von großer Bedeutung, sondern auch die Sicherheit des verwendeten Roboter-Agenten.
Das zweite Messkriterium ist die *Stoßzahl*, $Mess = SZ$, die der Anzahl der stattgefundenen Kollisionen entspricht. Dieses Messkriterium wird nur für das Navigationsproblem in Anspruch genommen und ist für die "Überlebenschance" des echten Roboters wichtig. Es wird ähnlich wie der Lernfortschritt bestimmt. Die über E Episoden gemessenen Kollisionen werden für jedes Element der Stichprobe bestimmt. Der Wert für jedes Element der Stichprobe wird über N Versuche noch einmal gemittelt. Dann wird die Varianz v^{SZ} berechnet. Die Unterschiede der Stoßzahl werden für unterschiedliche Konfigurationen mithilfe des t-Tests geprüft.

3.7 Wahl der Codierungsmethode

An dieser Stelle werden die ersten Ergebnisse, die zur Wahl der geeigneten Codierungsmethode des Zustandsraumes für das MCP und das Navigationsproblem angewendet werden, dargestellt.

3.7.1 Codierung im Mountain-Car-Problem

Zuerst werden die unterschiedlichen Codierungsmethoden für das Mountain-Car-Problem (MCP) untersucht. Der Zustandsraum ist ein 2D-Teilraum $[-1.2, 0.6] \times [-0.05, 0.05]$ mit zwei Komponenten, der Position $x \in [-1.2, 0.6]$ und Geschwindigkeit $\dot{x} \in [-0.05, 0.05]$ des Agenten. Der Zustandsraum wird in 11×11 Teile aufgeteilt. Die Basisfunktionen werden gleichmäßig über den Teilraum verteilt. Bei Tile-Coding (TC) wird nur ein Tiling verwendet.
Die eingeführten Codierungsmethoden werden miteinander verglichen. Insgesamt werden die Versuche für vier unterschiedliche Konfigurationen durchgeführt, vergleiche Tabellen 3.1 und 3.2. Der Unterschied beider Tabellen liegt in der Einbindung der Eligibility-Traces. In Tabelle 3.1 kommen die Eligibility-Traces nicht zum Einsatz ($\lambda = 0$). Dagegen haben die Eligibility-Traces in Tabelle 3.2 eine große Auswirkung ($\lambda = 0.95$). Für drei von vier Konfigurationen wird die Nutzenfunktion mit einer linearen Funktion approximiert. Diese drei Konfigurationen unterscheiden sich in der Codierungsmethode für den Zustandsraum, Tile-Coding-Methode (in den Tabellen als TC bezeichnet), Coarse-Coding-Methode (in den Tabellen als CC bezeichnet) und Codierung mit RBF (in den Tabellen als RBF bezeichnet). In der letzten Konfiguration wird die Nutzenfunktion durch ein "echtes" RBF-Netzwerk approximiert, in der Tabelle wird diese Konfiguration als RBF-Netzwerk bezeichnet. In diesem Fall werden die Parameter der Ausgabeschicht und der verdeckten Schicht gleichzeitig im Onlinebetrieb gelernt, dabei wird der zusätzliche Lernparameter auf den Wert $\alpha_\mu = 10^{-5}$ gesetzt. Die Ergebnisse zum Einfluss dieser zusätzlichen Lernparameter auf den Lernprozess werden weiter unten in Unterkapitel 3.7.2 beschrieben.
Die Anwendung der CC-Methode für das MCP erweist sich im Vergleich zum Tile-Coding mit nur einem Tiling als besser geeignet. Der Vergleich ohne Einbindung der Eligibility-Traces zeigt eine deutliche Verbesserung der Lerngeschwindigkeit von bis zu 45% mit der CC-Methode im Vergleich zum Verfahren mit der Tile-Coding-Methode, vergleiche erste, zweite und vierte Spalte in Tabelle 3.1. Die Codierung mit RBF ohne Anwendung der Eligibility-Traces ist im Vergleich zur Codierung mit Tile-Coding um bis zu 47.9% vorteilhafter, siehe dritte Zeile und vierte Spalte. Als beste Konfiguration stellt sich in diesem Fall die Approximation der Nutzenfunktion durch ein "echtes" RBF-Netzwerk mit Adaption der Parmeter in beiden Schichten des RBF-Netzwerks heraus, vergleiche vierte Spalte in der Tabelle. Diese Konfiguration besitzt aber eine größere Varianz als Coarse-Coding und Codierung mit RBF, vergleiche dritte Spalte in der Tabelle.
Wenn die Eligibility-Traces zum Einsatz kommen, wird der Unterschied zwischen allen Konfigurationen

Tabelle 3.1: Vergleich der Codierungsmethoden ohne Einbindung der Eligibility-Traces ($\lambda = 0$) für das MCP, Anzahl der Teile bzw. Zentren = 11×11. Das Tile-Coding enthält ein Tiling. Verbesserung der Lerngeschwindigkeit im Vergleich zum Tile-Coding. $\alpha = 0.5, \epsilon = 0, \gamma = 1.0$, max. Anzahl der Schritte = 10000, Anzahl der Episoden = 250, Parameter für modifizierte RBF $\alpha_\mu = 10^{-5}$

Methode	Lerngeschwindigkeit	Varianz	Verbesserung in %	t-Test LP
TC	368.1	38.1	0%	0
CC	202.3	4.9	45.0%	1
RBF	191.9	9.0	47.9%	1
RBF-Netzwerk	175.9	10.0	52.2%	1

Tabelle 3.2: Vergleich der Codierungsmethoden mit Einbindung der Eligibility-Traces ($\lambda = 0.95$) für das MCP, Anzahl der Teile bzw. Zentren = 11×11. Das Tile-Coding enthält nur ein Tiling. Verbesserung der Lerngeschwindigkeit im Vergleich zum Tile-Coding. $\alpha = 0.2, \epsilon = 0, \gamma = 1.0$, max. Anzahl der Schritte = 10000, Anzahl der Episoden = 250, Parameter für modifizierte RBF $\alpha_\mu = 0.0001$

Methode	Lerngeschwindigkeit	Varianz	Verbesserung in %	t-Test LP
TC	198.1	19.9	0%	0
CC	168.7	10.2	14.8%	1
RBF	173.5	12.8	12.4%	1
RBF-Netzwerk	190.4	11.7	4.0%	1

deutlich kleiner. Die Ergebnisse für diesen Fall sind in Tabelle 3.2 zusammengefasst. Die CC-Methode erzielt eine Verbesserung von bis zu 14.8% im Vergleich zur TC-Methode, siehe Zeile 2. Die Codierung mit RBF erzielt eine Verbesserung von bis zu 12.4%, siehe Zeile 3. Bei Anwendung eines "echten" RBF-Netzwerks wird eine Verbesserung von nur 3.9% gemessen, siehe 4. Spalte in der Tabelle. Die Einbindung der Eligibility-Traces hat entscheidend zur Beschleunigung des Lernens beigetragen. An den Werten der Lerngeschwindigkeit, siehe 2. Spalte in beiden Tabellen, kann beobachtet werden, dass die Einbindung der Eligibility-Traces eine Verbesserung von bis zu 40% für das Verfahren mit Tile-Coding hervorruft. Für die beiden anderen Codierungsarten (Coarse-Coding und RBF-Coding) bringt diese Erweiterung eine Verbesserung von nur höchstens 15% für Coarse-Coding und bis zu 10% für die RBF-Codierung. Dabei liefert die Einbindung der Eligibility-Traces für das Verfahren mit Codierung der Nutzenfunktion als "echtes" RBF-Netzwerk mit Adaption der verdeckten Schicht sogar eine Verschlechterung, siehe 2. Spalte in beiden Tabellen. Die Lerngeschwindigkeit liegt für modifizierte Verfahren bei 175.9 mit ausgeschalteten Eligibility-Traces ($\lambda = 0$) im Vergleich zu 190.38 für $\lambda = 0.95$. Diese Beobachtung kann so erklärt werden, dass am Anfang eine Approximation der Nutzenfunktion an die sehr ungenauen Messungen stattfindet. Der Prozess der Eligibility-Traces verstärkt die Anpassung an die fehlerhaften Trainingsdaten, so dass eine lange Zeit benötigt wird um diese Unstimmigkeiten zu beseitigen. In Abbildung 3.6 sind fünf Lernkurven dargestellt. Eine Lernkurve stellt die Abhängigkeit der über 100 Versuche gemittelten Anzahl der Schritte bis zum Ziel von der Anzahl der durchgeführten Episoden dar. Mithilfe einer Lernkurve kann die Entwicklung des Lernvorgangs veranschaulicht werden. Zwei Lernkurven stellen den Lernprozess mit "echten" RBF-Netzwerken dar. Die beiden Lernkurven unterscheiden sich im Einfluss der Eligibility-Traces ($\lambda = \{0.95, 0\}$). Diese Lernkurven sind als RBF, $\lambda = \{0.95, 0.0\}$, $\alpha = 0.2$, $\alpha_\mu = 0.00001$ in der Legende bezeichnet. Zwei weitere Lernkurven stellen den Lernprozess mit der Codierung mit RBF dar, sie unterscheiden sich auch im Einfluss der Eligibility-Traces

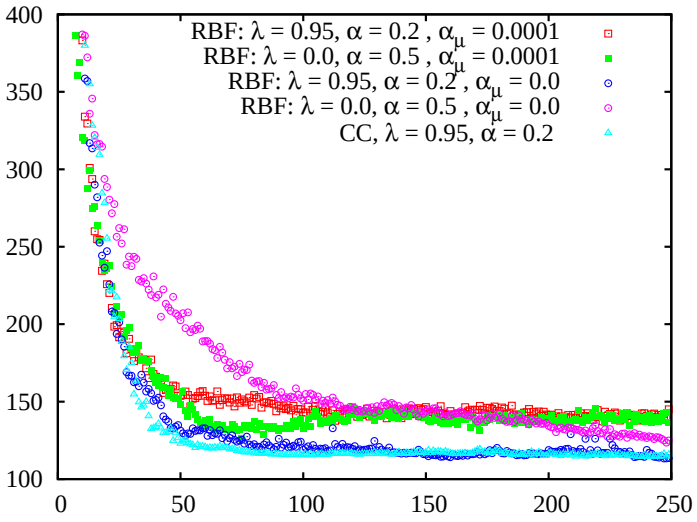


Abbildung 3.6: Lernverlauf für unterschiedliche Parametrisierungs- und Codierungsverfahren. RBF-Codierung mit Modifikation und Auswirkung der Eligibility-Traces auf den Lernvorgang. Zum Vergleich wird die Lernkurve für die Parametrisierung mit den besten Ergebnissen für das Coarse-Coding dargestellt.

($\lambda = \{0.95, 0\}$) und sind in der Legende als RBF, $\lambda = \{0.95, 0.0\}$, $\alpha = 0.2$, $\alpha_\mu = 0$ gekennzeichnet. Die fünfte Lernkurve stellt den Lernprozess mit Coarse-Coding und Eligibility-Traces dar und ist als *CC*, $\lambda = 0.95, \alpha = 0.2$ gekennzeichnet. Die dargestellten Lernkurven haben ein unterschiedliches Konvergenzniveau. Die Lernkurven mit Codierung mit RBF bzw. CC mit Einfluss der Eligibility-Traces zeigen den besten Lernverlauf und erreichen am schnellsten eine sehr gute Anzahl der bis zum Ziel benötigten Schritte des MCP-Agenten (ungefähr 115). Der Lernverlauf der Verfahren mit Codierung der RBF ohne Einfluss der Eligibility-Traces ist etwas langsamer, am Ende des Lernprozesses (250. Episode) nähert sich diese Kurve aber den beiden besten Lernkurven an. Für die Approximation der Bewertungsfunktion mit ”echtem” RBF-Netzwerk wird eine genau umgekehrte Beobachtung gemacht. Der Lernprozess ohne ist schneller als mit Einfluss der Eligibility-Traces. Die TC-Methode ist nicht optimal für das Mountain-Car-Problem. Diese Codierungsmethode wird aber in den meisten Versuchen für das MCP angewendet, da sie auch oft in der Literatur zum Einsatz kommt.

3.7.2 Parameteranalyse der verdeckten Schicht des RBF-Netzwerks

An dieser Stelle wird der Einfluss des zusätzlichen Lernparameters α_μ, der durch Anwendung des ”echten” RBF-Netzwerks zum Einsatz kommt, untersucht, vergleiche Lernregel für den Parameter μ in Gleichung (3.18). Der Lernprozess findet in der Ausgabeschicht und der verdeckten Schicht statt. Die Untersuchungen finden im MCP statt. Zu bemerken ist, dass die beim Start des SARSA-Algorithmus erzielten Werte für die Adaption der Q-Funktion[17] keine guten Trainingsdaten sind, so dass eine zusätzliche Adaption der inneren Parameter des RBF-Netzwerks auf Grundlage von sehr ungenauen Trainingsdaten nicht sinnvoll ist.
In Tabelle 3.3 ist die relative Verbesserung der Lerngeschwindigkeit des Lernprozesses angegeben, siehe zweite Spalte ”Verbesserung in %”. Der positive Wert bedeutet eine Verbesserung der Lerngeschwindigkeit. Der negative Wert bedeutet eine Verschlechterung der Lerngeschwindigkeit. Der Vergleich wird mit

[17] Die Q-Funktion ist mit Nullen oder Zufallswerten vorinitialisiert.

dem Lernen ohne zusätzliche Adaption der Parameter in der verdeckten Schicht des RBF-Netzwerks, also $\alpha_\mu = 0$, angestellt. Der Lernparameter α_μ steuert die Stärke des Einflusses auf die Adaption des Parameters der RBF. Dieser Parameter wird variiert, vergleiche erste Zeile der Tabelle. In der dritten Zeile wird vorgegeben, ob die erzielte Verbesserung bzw. Verschlechterung signifikant ist.
Aus der Tabelle wird ersichtlich, dass die Lernschritte (3.19), (3.18) für RBF-Netzwerke, siehe Abbildung 3.3, eine Beschleunigung des Lernprozesses aufweisen können, wenn der Lernparameter α_μ richtig eingestellt ist. Die Einbindung der Eligibility-Traces mit einem zusätzlichen Justierungsschritt in der verdeckten Schicht ist nicht ratsam, vergleiche die Ergebnisse in Unterkapitel 3.7.1.
In Tabelle 3.3 lässt sich der störende Einfluss dieser Adaption von bis zu -1.8% bei $\alpha_\mu = 0.001$ beobachten. Aber auch ein positiver Einfluss auf die Lerngeschwindigkeit von bis zu 8.3% für eine kleine Lernschrittweite von $\alpha_\mu = 1e-005$ wird beobachtet. Dieser kleine Wert für die optimale Lernschrittweite ist aber nicht überraschend, da für das stochastische Gradientenabstiegsverfahren in der Regel kleinere Lernschrittweiten empfohlen werden, um eine Annäherung an das eigentliche Gradientenabstiegsverfahren zu erhalten [Zell, 2003, Kap. 8, S. 114]. Zum Beispiel wird in [Menache et al., 2005, Kap. 5.2, S. 17] der Wert für die Lernschrittweite auf $5 \cdot 1e - 005$ gesetzt, was auch dem erzielten optimalen Wert im durchgeführten Versuch ähnelt. Der Lernschritt wird für einen besseren Vergleich mit beiden Vorzeichen durchgeführt, vergleiche Tabelle 3.3.

Tabelle 3.3: Relative Verbesserung im Vergleich zur RBF-Codierung ohne Anpassung der verdeckten Schicht. α_μ ist der Lernparameter für die Anpassung der verdeckten Schicht des RBF-Netzwerks, siehe Gleichung (3.18), Anzahl der Teile im Tiling = 11×11, $\alpha = 0.5, \lambda = 0, \epsilon = 0, \gamma = 1.0$, max. Anzahl der Schritte = 10000, Anzahl der Episoden = 250

α_μ	Verbesserung in %	t-Test LP
-0.0001	-38.7%	1
0.01	3.7%	1
0.001	-1.8%	0
0.0001	7.0%	1
1e-005	8.3%	1
1e-006	1.1%	0
1e-007	3.6%	1
1e-008	-1.3%	0
1e-009	1.2%	0

Die Abhängigkeit der Lernparameter α_μ von der Verbesserung bildet eine Parabel mit einem klaren Maximum in der Mitte der untersuchten Parameterskala $\alpha_\mu = 1e - 005$. Diese Tatsache deutet darauf hin, dass dieses Verfahren eine feine Kalibrierung des Parameters α_μ benötigt. Die Werte 0.01 und 0.001 sind noch zu groß, siehe Tabelle 3.3. Werte für $\alpha_\mu < 1e - 006$ sind schon zu klein, um eine signifikante Auswirkung der Anpassung der Verteilung der RBF zu messen. Der Wert $\alpha_\mu = 1e - 005$ ist fein genug, um eine Annäherung an das eigentliche Gradientenabstiegsverfahren darzustellen, und noch groß genug, um eine Verbesserung der Ergebnisse zu erzielen.
Die Abbildungen 3.7 und 3.8 dienen zur Veranschaulichung der Auswirkung des Anpassungsschritts in der verdeckten Schicht des RBF-Netzwerks. In beiden Abbildungen ist die Landschaft des Basisvektors dargestellt. Diese Landschaft ist durch die Aufsummierung aller Basisfunktionen $\Sigma() = \sum_{i=1}^{N} \phi_i()$ entstanden. Diese Funktion $\Sigma((x, \dot{x}))$ ist für jeden Zustand des Zustandsraumes $(x, \dot{x}) \in [-1.2, 0.6] \times [-0.05, 0.05]$ abgebildet.
In Abbildung 3.7 ist die Landschaft mit den justierten RBF im RBF-Netzwerk abgebildet. In Abbildung

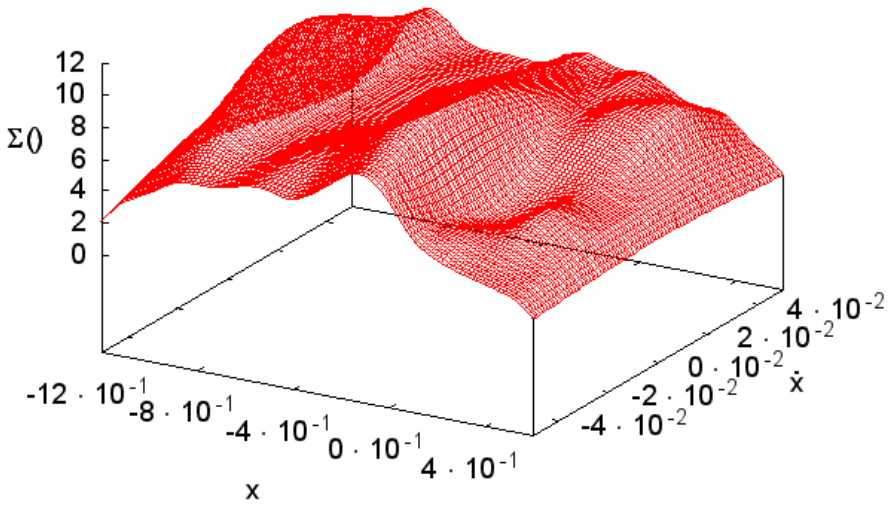


Abbildung 3.7: Landschaft des Basisvektors für eine justierte Verteilung des Merkmalsets über den Zustandsraum für das MCP. Diese Landschaft ist durch die Aufsummierung aller Basisfunktionen $\Sigma() = \sum_{i=1}^{N} \phi_i()$ entstanden.

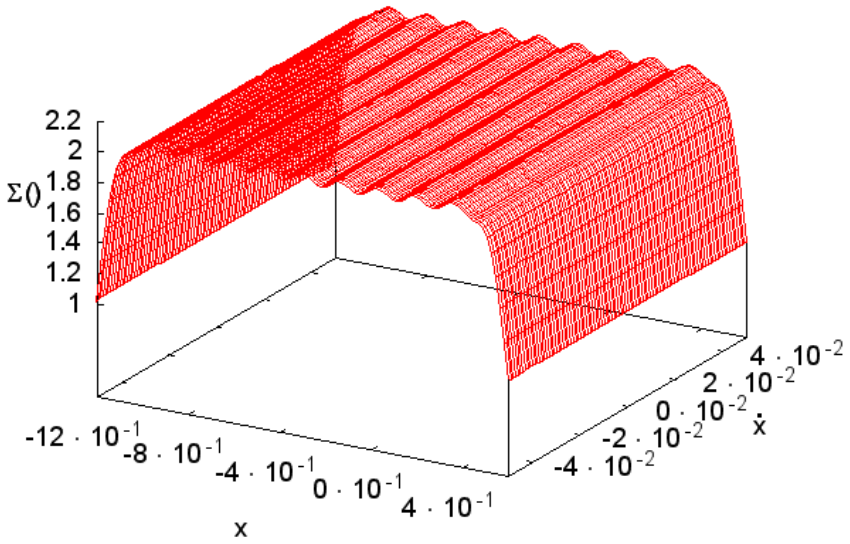


Abbildung 3.8: Landschaft des Basisvektors für eine gleichmäßige Verteilung des Merkmalsets über den Zustandsraum für das MCP. Diese Landschaft ist durch die Aufsummierung aller Basisfunktionen $\Sigma() = \sum_{i=1}^{N} \phi_i()$ entstanden.

3.8 ist die Landschaft mit über den Zustandsraum gleichverteilten RBF dargestellt. Diese Landschaften sind nach einem Lernprozess von 250 Episoden entstanden. Jede dieser Episoden beinhaltet im Durchschnitt 175 Lernschritte für ein RBF-Netzwerk mit Anpassungsschritt ($\alpha_\mu = 1e - 005$) und 191 Lernschritte für die Codierung mit RBF ohne Anpassungsschritt ($\alpha_\mu = 0$) , vergleiche auch Tabelle 3.1. Also werden für das RBF-Netzwerk 250×175 Adaptionsschritte für die Parameter in der verdeckten Schicht

durchgeführt.
Für den Algorithmus mit zusätzlicher Justierung der RBF über den Zustandsraum ist eine Landschaft mit vielen Unregelmäßigkeiten, vielen Maxima und Minima, zu beobachten. Diese Landschaft bedeutet, dass die unterschiedlichen Zustände im Zustandsraum einen unterschiedlichen Einfluss auf die Bildung der approximierten Nutzenfunktion haben.
Die in Abbildung 3.8 dargestellte Landschaft besteht aus einer fast flachen Ebene. Das bedeutet, dass alle Zustände die gleiche Auswirkung auf die Bildung der approximierten Nutzenfunktion haben.
Die lineare Approximation der Nutzenfunktion mit einer der Standard-Codierungsmethoden wie Tile-Coding, Coarse-Coding und Codierung mit RBF mit im ganzen Zustandsraum gleichverteilten Merkmalen reichen für die betrachtete Aufgabenstellungen, das MCP und das Navigationsproblem, aus. Ohne den Zusatzschritt des Nachjustierens der Position der Basisfunktionen werden auch gute, in den meisten Fällen sogar bessere Ergebnisse erzielt. In der Realisierung sind die gleichverteilten Merkmale über den Zustandsraum aber einfacher, was weniger potenzielle Fehler verursachen kann.

3.7.3 Codierung im Navigationsproblem

Für einen kontinuierlichen Zustandsraum im Navigationsproblem werden unterschiedliche Arten der Codierung miteinander verglichen. Die Versuche werden für Umgebung 1, siehe Abbildung 4.18(a), mit einem Tiling von $10 \times 10 \times 24$ durchgeführt. In Umgebung 1 muss der Agent keine komplexen Manöver erlernen, um das Ziel zu erreichen. Der Zustandsraum des Roboter-Agenten besteht aus drei Komponenten, der Position des Agenten in der befahrbaren Ebene und seiner Orientierung, vergleiche Unterkapitel 2.5.1. Der Versuch wird mit dem Einfluss der Eligibility-Traces ($\lambda = 0.9$) gestartet, da sich die Einbindung der Eligibility-Traces im MCP als sinnvoll herausgestellt hat. Die Parametrisierung während des Versuchs ist: $\alpha = 0.2, \epsilon = 0.1, \gamma = 0.95, \lambda = 0.9$.
Die Untersuchungen zeigen, dass die Tile-Coding-Methode für das Navigationsproblem gut geeignet ist,

Tabelle 3.4: Verschlechterung des Lernfortschritts relativ zur Tile-Coding-Methode und Signifikanz der Aussagen für das Navigationsproblem in Umgebung 1, siehe Abbildung 4.18(a), mit einem Tiling von $10 \times 10 \times 24$. Parametrisierung während des Versuchs: $\alpha = 0.2, \epsilon = 0.1, \gamma = 0.95, \lambda = 0.9$, max. Anzahl der Schritte = 50000, Anzahl der Episoden = 250.

Methode	Verbesserung in %	t-Test LP
Coarse Coding	-18.93%	1
RBF	-15.52%	1

siehe Tabelle 3.4. Der Agent muss keine komplexen Manöver durchführen, so dass bereits eine grobe Diskretisierung des Zustandsraumes die gewünschte Lernqualität ermöglicht. Bei der Coarse-Coding-Methode wird eine Verschlechterung von bis zu 19% gemessen, bei der RBF-Diskretisierungsmethode eine Verschlecherung von bis zu 16%, siehe Zeile 1 und 2 in der Tabelle. Die CC-Methode und die Codierung mit RBF erlauben eine feinere Diskretisierung des Zustandsraumes. Für das MCP ist die feinere Diskretisierung vorteilhafter. Aus diesem Grund ist der Lernprozess bei Anwendung der CC-Methode bzw. RBF-Methode schneller, vergleiche Tabelle 3.2. Für das Navigationsproblem ist eine feine Diskretisierung des Zustandsraumes nicht notwendig, mit Tile-Coding werden sehr gute Ergebnisse erzielt.
In der vorliegenden Arbeit wird für die meisten Versuche Tile-Coding als Codierungsmethode verwendet. Die Versuche, in denen die CC-Methode statt der TC-Methode angewendet wird, werden explizit

gekennzeichnet.

3.8 Zwischenstand

In diesem Abschnitt wurden die Grundlagen des Reinforcement-Learning-Ansatzes eingeführt. Das endliche MDP, sowie die im MDP definierte Bewertungsfunktion, auf deren Grundlage die benötigten Entscheidungen über die auszuführenden Aktionen getroffen werden können, wurden eingeführt.
Nur wenige Arten von Problemen können als ein endliches MDP beschrieben werden. Die Probleme mit einem kontinuierlichen Zustandsraum können durch den Diskretisierungsschritt zu Problemen im endlichen MDP projiziert werden. Es existieren zahlreiche Realisierungen des Diskretisierungsschrittes. In diesem Abschnitt wurden drei unterschiedliche Methoden zur Diskretisierung des Zustandsraumes eingeführt. Diese drei Codierungsmethoden, sowie die lineare Approximation der Nutzenfunktion sind zum größten Teil dem Buch von Sutton [Sutton & Barto, 1998] entnommen. In diesem Abschnitt wurde auch die Möglichkeit der Approximation der Nutzenfunktion durch ein ”echtes” RBF-Netzwerk dargestellt. Die benötigten Lernschritte für die Parameter der verdeckten Schicht werden in der vorliegenden Arbeit hergeleitet, so dass die Parameteradaption durch die Erweiterung des SARSA-Algorithmus stattfindet.
In diesem Abschnitt wurden bereits die ersten Ergebnisse präsentiert. Das verfolgte Ziel bei der Ermittlung der Ergebnisse war die Möglichkeit die eingeführten Codierungsmethoden miteinander zu vergleichen, um schon an dieser Stelle eine Vorstellung über die Vorteile der einzelnen Methoden zu erlangen.
Der Teil mit der Beschreibung der grundlegenden Begriffe und Methoden ist an dieser Stelle abgeschlossen. Die folgenden Kapitel werden der Erweiterung der Lernmethoden und Entwicklung der lernbasierten, rationalen Agenten zur Lösung der eingeführten Probleme gewidmet.

Kapitel 4

Heuristisch erweitertes Reinforcement-Learning

Dieses Kapitel ist dem Thema Effizienzsteigerung durch Erweiterung des Lernelements von rationalen Agenten gewidmet, siehe Abbildung 1.1. Die vorgeschlagene Effizienzsteigerung wird durch die Erweiterung des in Kapitel 3.4.6 eingeführten Gradienten-SARSA(λ)-Algorithmus um heuristisches Wissen erzielt. Im vorliegenden Kapitel werden erste Überlegungen, unter welchen Rahmenbedingungen die Konvergenz des Verfahrens zu einer guten Lösung durch die vorgenommenen Erweiterungen nie verletzt wird, eingeführt. Die dargestellten Herleitungen, die für das konkrete Navigationsproblem durchgeführt worden sind, sind zwar nicht vollständig, bilden aber ein Fundament. Die erzielten theoretischen Ergebnisse werden durch praktische Untersuchungen bestätigt, so dass die vorgegebene Richtung der durchgeführten Herleitung ihre Bestätigung findet. Die Auswirkung der vorgeschlagenen Erweiterung des Lernalgorithmus auf die Steigerung der Lerngeschwindigkeit wird für unterschiedliche Problemstellungen gemessen.
Heuristisches Wissen kann a-priori, rudimentäres, verstecktes Kontextwissen sein. In der vorliegenden Arbeit wird das heuristische Wissen in Form einer heuristischen Funktion dargestellt. Je nach Problemstellung und vorhandenem Wissen kann eine solche heuristische Funktion unterschiedlich definiert sein. Die Definitionen der unterschiedlichen heuristischen Funktionen werden im vorliegenden Kapitel präsentiert und stellen nur Beispiele dar. Dabei wird unterschieden, wie die heuristische Funktion in den Reinforcement-Learning-Ansatz eingebunden wird. Die erste Möglichkeit entsteht durch das Einbinden der heuristischen Funktion in der Aktionswahl. In diesem Fall wird die ϵ-gierige Strategie um den heuristischen Anteil erweitert und die erweiterte Strategie als ϵ^*-gierige Strategie bezeichnet[18]. Die zweite Möglichkeit entsteht durch Einbindung der heuristischen Funktion in den Lernprozess selbst. Diese erweiterte Methode wird später als SARSA*-Algorithmus bezeichnet.
In Kapitel 4.1 wird die Grundidee, die hinter der vorgeschlagenen Erweiterung steckt, präsentiert. Zwei Möglichkeiten, wie die vorgeschlagene Erweiterung realisiert werden kann und abschließend ein kurzer Überblick über die auf diesem Gebiet bereits existierenden Arbeiten werden vorgestellt.
In Kapitel 4.2 werden wichtige Begriffe, die in Zusammenhang mit der vorgenommenen Erweiterung zur Geltung kommen, eingeführt und mithilfe zahlreicher Beispiele erläutert.
Die unterstützenden Herleitungen des zulässigen Intervalls für die heuristischen Parameter werden in den Kapiteln 4.3 für die ϵ^*-gierige Strategie und 4.4 für den SARSA*-Algorithmus durchgeführt. Die beiden Kapitel sind für das Verständnis des Grundkonzepts irrelevant und stellen nur den Weg, mit dem die im vorherigen Kapitel zusammengefassten Vorschläge für das zulässige Intervall erzielt worden sind, dar. Die erzielten Ergebnisse werden mit den Ergebnissen, die mit einer vollständigen Suche ermittelt worden sind, in beiden Kapiteln verglichen.
Das um eine heuristische Funktion erweiterte Reinforcement-Learning (RL) wird hinsichtlich der erziel-

[18] Diese Bezeichnung stammt aus der Assoziation mit dem A^*-Algorithmus.

ten Beschleunigung der Lerngeschwindigkeit analysiert. Dabei werden die folgenden, unterschiedlichen Problemstellungen verwendet, um eine hohe Aussagekraft der erreichten Beschleunigungen zu erzielen: das für das endliche MDP definierte in [Sutton & Barto, 1998, Kap. 3.7, S. 71] eingeführte Gitterweltproblem (engl.: gridworld problem)[19], das in Kapitel 2.3 eingeführte Mountain-Car-Problem (MCP) und das in Kapitel 2.4 eingeführte Navigationsproblem, formuliert für ein unendliches MDP. Die ersten Ergebnisse für das elementare Gitterweltproblem werden in Kapitel 4.5 präsentiert. Die weitere Analyse der vorgeschlagenen Erweiterungen des Lernprozesses beinhaltet die Ergebnisse in Kapitel 4.6 für das MCP. Der Abschnitt wird mit Ergebnissen für das Navigationsproblem aus Kapitel 4.7 abgeschlossen.

4.1 Heuristisch erweitertes RL

In diesem Kapitel wird zuerst die grobe Idee skizziert und anschließend werden zwei Methoden vorgestellt, wie die heuristische Funktion in den Agenten eingebaut werden kann. Die beschriebene Idee zur Beschleunigung des Lernprozesses wird durch die Einbindung der heuristischen Funktion in den Lernprozess oder in der Aktionswahl (bei den RL-Methoden) realisiert.

4.1.1 Grundidee

Der große Nachteil der RL-Algorithmen ist ihre langsame Konvergenz. Sie resultiert aus dem Ausführen und Ausprobieren von Zuständen und Aktionen, die für das Erreichen des Ziels unnötig sind. Je feiner das Gitter des Zustandsraums gewählt wird, desto mehr Kombinationen aus Zuständen und Aktionen müssen untersucht werden, um die gesuchte optimale Lösung zu finden, und desto langsamer ist die Konvergenz zur optimalen Lösung. Dieses Problem ist auch unter dem Begriff ”Fluch der Dimensionalität” bekannt.
Wenn dem Algorithmus zusätzlich die Bedeutung (Priorität) der Zustände mitgeteilt wird, also welche Zustände für das gestellte Problem am wichtigsten sind, ist zu erwarten, dass die gesuchte optimale Lösung schneller gefunden werden kann, also der Lernprozess positiv beeinflusst werden kann.
Dem Reinforcement-Learning-Ansatz werden diese Prioritäten als heuristisches Wissen vorgegeben. Heuristisches Wissen kann aus dem Kontext der Problemstellung oder während des Lernprozesses gewonnen werden und wird in Form einer heuristischen Funktion eingebunden. Die heuristische Funktion ist eine Art einfache Vermutung für die Lösung der vorgegebenen Problemstellung, so dass die auf dem direkten Weg zum Zielpunkt liegenden Zustände eine bessere Güte haben als ihre Nachbarn. Diese Schätzung der Güte der Zustände stimmt nicht überein mit der tatsächlich gesuchten optimalen Strategie, kann aber hilfreich sein. Die heuristische Funktion beinhaltet die Bedeutung der Zustände für die Abwicklung der durchgeführten Mission. Die Bedeutung der Zustände kann zum Beispiel durch die Nähe zum Ziel gemessen werden. Wenn die wichtigen Zustände zuerst untersucht werden, ist zu erwarten, dass das Ziel schneller erreicht wird, und die optimale Lösung schneller bestimmt wird. Diese Schlussfolgerung gilt nicht immer und hängt vom Aufbau der Umgebung ab. Im Folgenden wird aber gezeigt, dass eine solche Vorgehensweise in durchschnittlichen Umgebungen zu höheren Lerngeschwindigkeiten führen kann.
Die Einbindung der heuristischen Funktion in der großen Klasse der Optimierungsprobleme hat ein solides Fundament. Die Idee der Anwendung der heuristischen Funktion im vorliegenden Kapitel ähnelt dem A^*-Algorithmus. In diesem Ansatz wird die Entscheidung über den nächsten untersuchten Knoten auf Grundlage von zwei Bestandteilen getroffen. Damit werden zuerst die Knoten untersucht, die

[19] Das Ziel im Gitterweltproblem liegt in der Bestimmung der optimalen Strategie, die den Agenten von jedem Zustand des Raumes in einer minimalen Anzahl an Schritten zum Zielzustand navigiert.

wahrscheinlich schnell zum Ziel führen. Der erste Teil der Kosten besteht aus den bisherigen Kosten, die angefangen beim Startknoten angefallen sind. Der zweite Teil ist die heuristische Kostenschätzung bis zum Ziel. Dies wird oft als Distanz bis zum Zielknoten (Luftlinie) definiert. In Analogie zum A^*-Algorithmus [Hart et al., 1968] wird das Kontextwissen auch in dieser Arbeit "heuristische Funktion" genannt.
Für die meisten Probleme, die mit RL-Algorithmen gelöst werden können, besteht die Möglichkeit eine heuristische Funktion ohne großen Aufwand zu definieren. Der Vorteil der RL-Algorithmen bleibt bestehen, da kein zusätzliches a-priori Wissen über das Modell der Umgebung benötigt wird. Zum Beispiel kann die Information im Navigationsproblem in den Lernprozess in Form einer heuristischen Funktion schon nach dem ersten Ankommen am Ziel eingebunden werden, auch wenn die Zielkoordinaten nicht von Anfang an vorhanden sind.

4.1.2 Heuristische Funktion in der Aktionswahl (ϵ^*-gierige Strategie)

Im SARSA-Algorithmus wird die Entscheidung über die nächste Aktion auf Grundlage der ϵ-gierigen Strategie getroffen, vergleiche z. B. Unterkapitel 3.3. Die erste Möglichkeit die Effizienz der Lernmethode zu erhöhen wird durch die Einbindung der heuristischen Funktion in die Entscheidung über die nächste Aktion in der aktuellen Situation realisiert. Mit diesem Schritt wird die vom Agenten verfolgte Strategie erweitert und die um eine heuristische Funktion erweiterte Strategie als ϵ^*-gierige Strategie bezeichnet. Die heuristische Funktion hat nur einen Einfluss auf die Bildung der Strategie.
Die durch die erweiterte Strategie getroffene Entscheidung über die nächste Aktion basiert auf zwei Bestandteilen, der Q-Funktion und der heuristischen Funktion, siehe Gleichung (4.1). Die Werte der heuristischen Funktion sind nicht im Lernschritt der Q-Funktion vertreten.

$$a = \widetilde{\pi}(s) \leftarrow \max_{a'}(Q(s,a') + h(\underbrace{\tau(s,a')}_{=s'})) \tag{4.1}$$

$\tau(s,a)$ ist eine Transitionsfunktion, siehe Gleichung (3.1).

4.1.3 Heuristische Funktion im Lernprozess (SARSA*-Algorithmus)

Eine alternative Möglichkeit die heuristische Funktion im RL-Ansatz zu berücksichtigen wird direkt in die Update-Regel (3.10) des Standard-SARSA-Algorithmus eingearbeitet. Für den erweiterten Ansatz (SARSA*-Algorithmus) entsteht der erweiterte Lernschritt, siehe Gleichung (4.2). Die heuristische Funktion bildet mit dem Belohnungsmodell eine Art stetiges Belohnungsmodell.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + h(s) + \gamma \cdot \max_{\widetilde{a}} Q(\underbrace{\tau(s,a)}_{s'}, \widetilde{a}) - Q(s,a)) \tag{4.2}$$

wobei $h(s)$ die heuristische Funktion darstellt und der Term $r(s,a) + h(s)$ das heuristisch erweiterte Belohnungsmodell repräsentiert.
Auch die Gleichung (4.3) stellt den erweiterten Lernschritt der Q-Funktion für den $SARSA$-Algorithmus dar.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + h(s') + \gamma \cdot \max_{\widetilde{a}} Q(\underbrace{\tau(s,a)}_{s'}, \widetilde{a}) - Q(s,a)) \tag{4.3}$$

Die beiden modifizierten Lernregeln (4.2) und (4.3) verfolgen das gleiche Prinzip der Einbindung der heuristischen Funktion in den Lernprozess. Allerdings ist bei Anwendung der Gleichung (4.3) im un-

endlichen MDP die Bestimmung des Folgezustands s' schwierig. Daher werden alle theoretischen Überlegungen basierend auf Gleichung (4.2) durchgeführt. Die theoretischen Überlegungen lassen sich aber auf Gleichung (4.2) übertragen. Für eine bessere Nachvollziehbarkeit der Auswirkung der heuristischen Funktion auf den Lernprozess sind die einzelnen Schritte des Standard-SARSA-Algorithmus sowie die einzelnen Schritte des SARSA*-Algorithmus für ein Problem im endlichen MDP in Anhang B beschrieben.

4.1.4 Literaturübersicht zur heuristischen Erweiterung des RL

Die beiden Ansätze sind bereits in einem Konferenzbeitrag der Autorin [Noglik et al., 2009] vorgestellt worden. Die theoretischen Überlegungen, unter welchen Bedingungen die Konvergenz der erweiterten Algorithmen gewährleistet werden kann, werden zum ersten Mal in der vorliegenden Arbeit vorgestellt.
Das Einbinden der heuristischen Funktion in die Aktionswahl wird auch in der Arbeit von [Bianchi et al., 2008] untersucht. In der Arbeit von Bianchi et al. ist aber ein Umgebungsmodell erforderlich. Das Modell wird während der ersten vom Agent durchgeführten Episode geschätzt. Und daraus wird die heuristische Funktion mithilfe des Gradientenverfahrens aufgestellt. Diese heuristische Funktion wird dann entsprechend der neuen Werte der Q-Funktion lediglich skaliert und nicht weiter adaptiert[20].
Eine ähnliche Idee des Einbindens einer stetigen Belohnung (heuristische Funktion im Lernprozess) ist zum Beispiel in [Papierok et al., 2008] zu finden. Dies ist ein Beitrag der Autorin zur Anwendung des RL-Algorithmus im Navigationsproblem. Der aufgestellte Ansatz ist aber auf die Entwicklung von reaktiven Fähigkeiten ausgerichtet und beinhaltet keine für die vorliegende Arbeit benötigte planende Fähigkeit.

4.2 Wichtige Begriffe zum heuristisch erweiterten RL

Die heuristische Funktion ist eine Art Bedeutungsmaß für jeden Zustand des Zustandsraums und kann in Konflikt zum eigentlichen Umgebungsmodell[21] stehen, d.h., dass die heuristische Aussage über die Bedeutung des Zustandes in Bezug auf das Erreichen des Ziels nicht der Wahrheit entspricht. An diesen Stellen können sogenannte *Zyklen* entstehen, in denen der Agent stecken bleiben kann. Wenn ein Zyklus entsteht, wird der erweiterte Lernalgorithmus nie zu einer Strategie konvergieren, die den Agenten in den entstandenen Zyklen zum Ziel steuert. Zyklen können vermieden werden, indem der Einflussparameter, der den im erweiterten RL heuristischen Einfluss steuert, richtig bestimmt wird.

4.2.1 Heuristische Funktion

Jeder Zustand wird im Vorfeld in Bezug auf das gestellte Problem mithilfe der heuristischen Funktion charakterisiert. Diese heuristische Funktion soll steuerbar und beschränkt sein. Der Wertebereich der heuristischen Funktion soll in Relation zum vorgegebenen Belohnungsmodell definiert werden, um die Lerngeschwindigkeit maximieren zu können. Die gewünschte Steuerung wird durch den heuristischen Einflussparameter realisiert.
Bei der Definition der heuristischen Funktion muss die folgende schwierige Frage beantwortet werden.

[20] In der vorliegenden Arbeit wird die heuristische Funktion nicht verändert, muss aber richtig parametrisiert sein. Die Bestimmung der Parameter wird weiter unten in Kapitel 4.3 beschrieben.
[21] Das Umgebungsmodell bleibt für den Agenten verborgen. Dieser Konflikt wird während des Lernprozesses zur Geltung kommen.

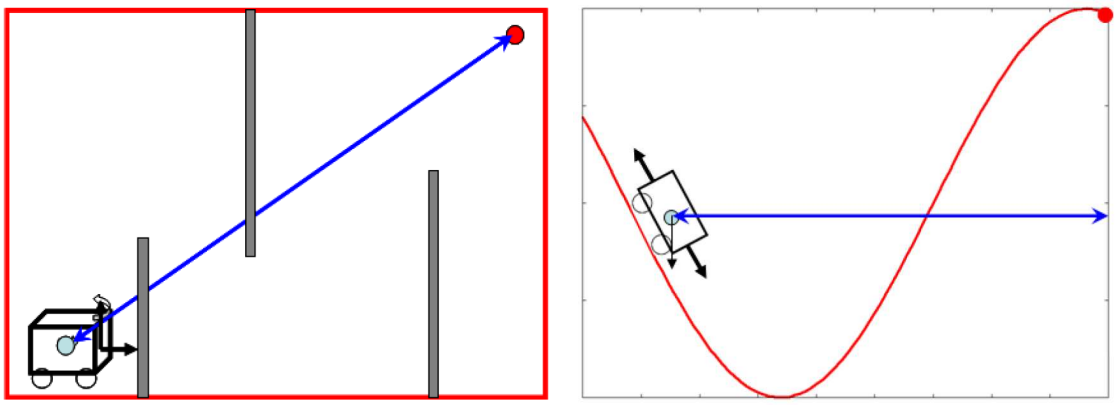
Welche Informationen können auf der Suche nach der optimalen Strategie hilfreich sein? Diese Fragestellung hängt von der zu lösenden Problemstellung ab. Zum Beispiel kann im Fall eines Navigationsproblems die Vermutung aufgestellt werden, dass sich der Agent in umso besserer Lage befindet, je näher er am Zielzustand ist. Es existiert keine allgemeingültige Definition der heuristischen Funktion. Die durch eine heuristische Funktion repräsentierten Informationen sollen in Abhängigkeit von der konkreten Aufgabenstellung und den verfolgten Zielen aufgestellt werden. An dieser Stelle wird eine beispielhafte Definition der heuristischen Funktion für das Navigationsproblem und das MCP eingeführt.
Eine mögliche Definition der heuristischen Funktion für das Navigationsproblem kann die Luftlinie von der Position des Agenten bis zum Zielzustand sein, siehe Abbildung 4.1(a). Eine so definierte heuristische Funktion liefert Informationen über die Nähe des Zieles ohne das Umgebungsmodell zu berücksichtigen. Die Heuristik lautet dabei, je näher das Ziel ist, desto besser ist die Lage des Agenten.

$$h(s) = k \cdot \underbrace{\frac{\sqrt{(s^{Ziel} - s)^T \cdot (s^{Ziel} - s)}}{Norm}}_{=:\widetilde{h}(s)} = k \cdot \widetilde{h}(s) \tag{4.4}$$

Der Einfluss der heuristischen Funktion auf den Lernprozess wird durch den Parameter k gesteuert. Die aufgestellte heuristische Funktion bleibt unverändert, muss aber entsprechend der gewählten Konfiguration des Standard-SARSA-Algorithmus kalibriert (richtig parametrisiert) werden. Der konstante Wert $Norm$ ist der Normierungsfaktor, der als maximale Distanz zwischen dem Ziel und dem Zustand aus dem Zustandsraum bestimmt werden kann. Dieser Wert kann ein Schätzwert sein.

$$Norm \leftarrow \max_{s \in S} \sqrt{(s^{Ziel} - s)^T \cdot (s^{Ziel} - s)}$$

Wenn die Zielposition s^{Ziel} am Anfang des Lernprozesses noch nicht bekannt ist, kann die heuristische Funktion, siehe Gleichung (4.4), erst nach dem ersten Erreichen des Zieles in den Lernprozess eingebunden werden.



(a) Navigationsproblem (b) Mountain-Car-Problem

Abbildung 4.1: Beispielprobleme mit möglicher Definition der heuristischen Funktion

Für das Mountain-Car-Problem wird zuerst die Heuristik ”Je näher die Position des Agenten an der Zielposition liegt, desto besser ist der gesamte Zustand des Agenten.” verfolgt. In Abbildung 4.1(b) wird diese Möglichkeit zur Definition der heuristischen Funktion mit dem blauen Pfeil angedeutet. Der Zu-

standsraum des Agenten für das MCP ist zweidimensional und besteht aus der Position und Geschwindigkeit des Agenten ($(x, \dot{x}) \in [-1.2; 0.6] \times [-0.07; 0.07]$, siehe auch Unterkapitel 2.3.1). In die Berechnung der heuristischen Funktion wird nur die Position des Agenten und die Nähe zu einer vorgegebenen Zielposition ($x^{Ziel} = 0.6$) eingebunden, da die Geschwindigkeit beim Erreichen des Ziels keine Bedeutung für den Agenten hat. Die so aufgestellte Heuristik ist nicht immer richtig bzw. aussagekräftig, kann aber hilfreich für den Lernprozess sein. Die Definition der heuristischen Funktion für das MCP sieht wie folgt aus:

$$h((x, \dot{x})) := h(x) = k \cdot \frac{\sqrt{(0.6 - x)^2}}{1.8} \tag{4.5}$$

Der Normierungsfaktor wird zu $Norm := \max_{x \in [-1.2;0.6]} |0.6 - x| := 1.8$ berechnet.
Um den heuristischen Einfluss im Lernprozess steuern zu können, wird in der Definition der heuristischen Funktion für das Navigationsproblem der heuristische Einflussparameter k eingeführt, siehe Gleichung (4.4) und (4.5). An dieser Stelle wird die Bedeutung dieses Parameters erläutert. Die heuristische Funktion repräsentiert je nach Vorzeichen des Parameters k eine unterschiedliche Prioritätenverteilung der Zustände über den Zustandsraum im Lernprozess.
Im Fall $k < 0$ sind die Werte der heuristischen Funktionen (4.4) und (4.5) in den Zuständen des Zustandsraums umso größer, je näher sich diese Zustände am Ziel befinden. Da die heuristische Funktion ohne Einbindung von Informationen über das Umgebungsmodell[22] definiert ist, kann diese Funktion auch eine ”falsche” Schätzung über die Priorität des Zustandes in Bezug auf das Erreichen des Ziels enthalten.
Im Fall $k > 0$ steht die als Distanz zum Zielzustand definierte heuristische Funktion eigentlich in Konflikt zur Problemstellung des betrachteten Navigationsproblems. In diesem Fall besagt die heuristische Funktion, dass je weiter sich ein Zustand vom Zielzustand befindet, desto größer ist der heuristische Wert dieses Zustandes. Eine so definierte heuristische Funktion kann dem Lernprozess aber auch einen Nutzen bringen, wenn die gesuchte optimale Strategie über den weitesten Zustand des Zustandsraumes verläuft. In der Regel ist es aber nicht ratsam die heuristische Funktion im Konflikt zur eigentlichen Problemstellung zu definieren. Dieser Fall wird der Vollständigkeit halber betrachtet.

4.2.2 Heuristische Funktion für endliche MDP

Die Verteilung des heuristischen Wissens über den Zustandsraum und somit die Bedeutung der heuristischen Funktion und des heuristischen Einflussparameters wird mithilfe des Gitterweltproblems für ein endliches MDP verdeutlicht. Mithilfe des Gitterweltproblems und der für das Problem konstruierten Szenarien wird eine bessere Nachvollziehbarkeit, Kontrollierbarkeit und Steuerbarkeit der durch die Einführung der Heuristik potenziell auftretenden Konflikte erreicht. Zwei Szenarien für das Gitterweltproblem sind so konstruiert, dass die angesprochenen Konflikte zwischen der Aussage der heuristischen Funktion und der tatsächlichen optimalen Strategie hervorgerufen werden. Die Anzahl der Zustände wird in beiden Szenarien klein gehalten. Das Gitterweltproblem ist ein vereinfachtes Navigationsproblem.

4.2.2.1 Gitterweltproblem

Das für das endliche MDP definierte Gitterweltproblem ist in [Sutton & Barto, 1998, Kap. 3.7, S. 71] eingeführt worden. Das Ziel des Agenten im Gitterweltproblem liegt im Erlernen der optimalen Strategie, die den Agenten von jedem Zustand des Raumes in einer minimalen Anzahl an Schritten zum vorgegebenen Zielzustand führt.
Der Aktionsraum im Gitterweltproblem besteht aus vier Aktionen. Der Agent kann sich nach links ($\leftarrow$),

[22] außer der Information über den Zielzustand

rechts ($\rightarrow$), oben ($\uparrow$) und unten ($\downarrow$) bewegen.
Wenn das Ausführen der Aktion nicht von einer Wand gestört wird, springt der Agent in den Nachbarzustand in die durch die Aktion vorgegebene Himmelsrichtung. Zum Beispiel wenn die Aktion nach links bewegen ($\leftarrow$) vom Zustand s_2 aus ausgeführt wird, landet der Agent im Zustand s_1, siehe Abbildung 4.2(a). Dieser Schritt wird wie folgt bezeichnet: $s_1 = \pi(s_2, \leftarrow)$. Eine Wand in der Umgebung wird durch hervorgehobene Linien in den Abbildungen für die Gitterweltszenarien dargestellt. Wenn eine Wand das Ausführen der Aktion stört, bleibt der Agent im alten Zustand. Zum Beispiel ist für die Umgebung in Abbildung 4.2(a) die Aktion links ($\leftarrow$) aus dem Zustand s_1 wegen der vorhandenen Wand nicht ausführbar, also $s_1 = \pi(s_1, \leftarrow)$. Für alle konstruierten Szenarien des Gitterweltproblems wird der Zielzustand in der Abbildung farblich gekennzeichnet, siehe Zustand s_0.
Das Belohnungsmodell im Gitterweltproblem ist wie folgt definiert. Für jeden Schritt bekommt der Agent eine Bestrafung von -1, es sei denn der Agent gelangt mit diesem Schritt zum Zielzustand, dann erhält der Agent keine Bestrafung. Dieses Belohnungsmodell wird hier als Standard-Belohnungsmodell bezeichnet. Im Belohnungsmodell mit Sonderbelohnung erhält der Agent eine Belohnung von $+100$, wenn er das Ziel erreicht. Im Belohnungsmodell mit Sonderbestrafung erhält der Agent eine Belohnung von -10, also eine Bestrafung, wenn eine Wand das Ausführen einer Aktion stört.

4.2.2.2 Heuristische Funktion im Gitterweltproblem

Die Umgebung ist in der Gitterwelt als eine $N \times M$-Matrix definiert, wobei alle Zustände $s_i, i = 0 \ldots s_{N \cdot M-1}$ in dieser Matrix so angeordnet sind, dass $s_i = s_{n \cdot M+m}$ für $n = 0, \ldots N-1, m = 0, \ldots M-1$, siehe Abbildung 4.2(a) mit $M = 4, N = 3$. Jeder Zustand wird durch einen zweidimensionalen Index $s_{n \cdot M+m} := (n, m)$ in der Matrix repräsentiert. Der Zustand mit maximalem Abstand vom Zielzustand s_0 hat den maximalen Index s_{11}. Die heuristische Funktion für das Navigationsproblem wird als direkte, normierte Distanz zum Zielzustand definiert. Diese Distanz wird für die oben eingeführte Indizierung als

$$\begin{aligned} d(s_i, s_{Ziel}) &= \sqrt{(s_{Ziel=n_{Ziel} \cdot M+m_{Ziel}} - s_{i=:n \cdot M+m})^T (s_{Ziel} - s_i)} \\ &= \sqrt{(n_{Ziel} - n)^2 + (m_{Ziel} - m)^2} \end{aligned} \tag{4.6}$$

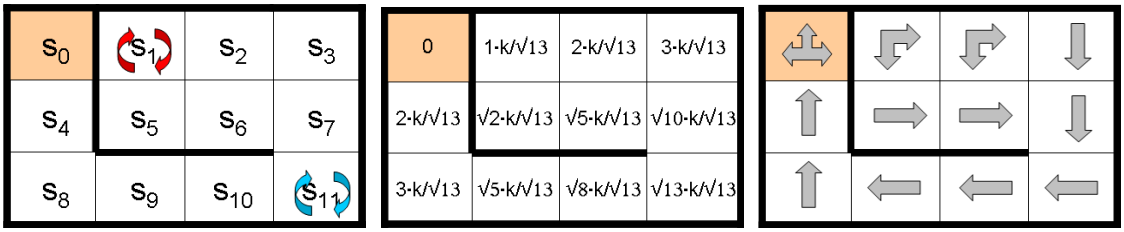
definiert. Die heuristische Funktion ist dann für jeden Zustand s_i mit Zielzustand s_0 wie folgt definiert:

$$h(s_i) = k \cdot \frac{d(s_i, s_0)}{d(s_0, s_{M \cdot N-1})}$$

4.2.2.3 Verwendete Szenarien

In Abbildung 4.2(a) ist das erste wichtige, im Gitterweltproblem verwendete Szenario abgebildet. Das Szenario ist so konstruiert, dass die angewendete Heuristik im Widerspruch zur gesuchten optimalen Strategie steht. Die Verteilung der Werte der aufgestellten heuristischen Funktion (4.6) für dieses Szenario ist in Abbildung 4.2(b) dargestellt. Die durch die heuristische Funktion repräsentierte Priorität der Zustände für den Lernprozess hängt vom Vorzeichen des heuristischen Einflussparameters k ab, siehe auch Unterkapitel 4.2.2.4.
In Abbildung 4.2(b) ist die gesuchte optimale Strategie des Agenten für das Szenario mit Weggabelung dargestellt. Diese Strategie ist für jeden Zustand aus dem Zustandsraum definiert, so dass der Agent von jedem Zustand s aus die minimale Anzahl an Schritten bis zum Erreichen des Ziels benötigt, wenn die abgebildete, optimale Strategie verfolgt wird. Die dargestellte Strategie ist die gesuchte optimale Lösung

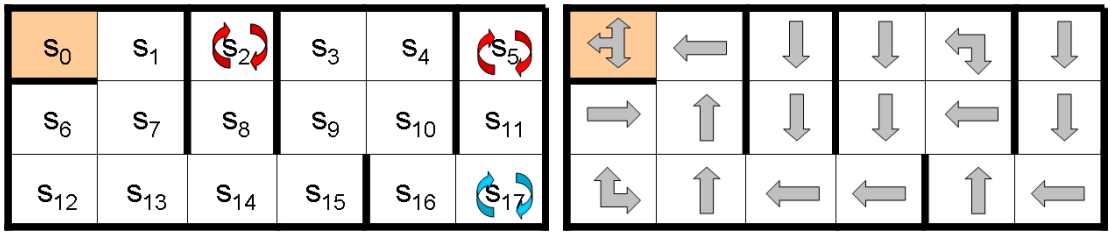
des Problems. Der Agent soll diese Strategie während des Lernprozesses erlernen. Das nächste einfache



(a) Modell (b) Verteilung der heuristischen Information (c) Optimale Strategie

Abbildung 4.2: Definition der heuristischen Funktion und der optimalen Strategie in der Umgebung mit Weggabelung. Bild (a): Umgebung mit durchnummerierten Zuständen. Mit roten und blauen Pfeilen werden die möglichen lokalen Extrempunkte, in Kapitel 4.2.3 als Zyklus-kandidaten eingeführt, gekennzeichnet. Bild (b): Heuristische Funktion ist die normierte Abstandsfunktion bis zum Ziel, mit $\sqrt{13} = \sqrt{3 \cdot 3 + 2 \cdot 2}$. Bild (c): Optimale Strategie, mithilfe des Standard-SARSA-Algorithmus generiert.

Szenario ”Umgebung mit Sackgassen” wird in Abbildung 4.3(a) aufgestellt, um die Ergebnisse aus der Umgebung mit Weggabelung zu bestätigen. Dieses Szenario ist auch so konstruiert, dass die heuristische Funktion in Konflikt zur optimalen Strategie steht.



(a) Modell (b) Optimale Strategie

Abbildung 4.3: Umgebung mit Sackgassen. Bild (a): Szenario mit durchnummerierten Zuständen, mit roten und blauen Pfeilen sind die vermutlichen lokalen Extrempunkte gekennzeichnet. Bild (b): Optimale Strategie, mithilfe des Standard-SARSA-Algorithmus generiert.

4.2.2.4 Deutung des heuristischen Einflussparameters k

Zum besseren Verständnis der Auswirkung des heuristischen Parameters wird an dieser Stelle die Auswirkung der heuristischen Funktion anhand des einfachen Gitterweltproblems näher betrachtet.
Für den Fall $k < 0$ besagt die heuristische Funktion im Gitterweltproblem für die Umgebung mit Weggabelung, dass zum Beispiel die Zustände s_1, s_5, s_4[23] am besten für das Erreichen des Zielzustands s_0 geeignet sind, da diese Zustände dem Ziel am nächsten liegen, siehe zum Beispiel Abbildung 4.2(a). In der Umgebung mit Weggabelung stimmt diese Aussage für den Zustand s_4 mit der Wirklichkeit überein. Die Zustände s_1 und s_5 sind aber vom Ziel durch eine Wand getrennt. Also stimmt die Heuristik mit der eigentlichen Umgebung an diesen Zuständen nicht überein. Dies gilt auch für die weiteren Zustände s_2, s_3, s_6, s_7. Somit steht die heuristische Funktion für dieses Problem an den Zuständen $s_1, s_2, s_3, s_5, s_6, s_7$ in Konflikt zur optimalen Strategie.

[23] An diesen Zuständen liefert die heuristische Funktion die kleinsten negativen Werte.

Für den Fall $k > 0$ sagt die heuristische Funktion, dass das Erreichen des weitesten Zustandes s_{11} den meisten Nutzen bringt, da $h(s_{11}) = k \cdot 1$. Somit steht eine so aufgestellte heuristische Funktion in Konflikt mit dem eigentlichen Zielzustand. Mit einer auf diese Weise definierten heuristischen Funktion und für richtig eingestellte Parameter $(k > 0, \gamma)$ lernt der Agent die optimale Strategie schneller. Dies lässt sich dadurch erklären, dass der Zustand s_{11} ein wichtiger Knoten der optimalen Strategie ist. Der Agent erlernt die optimale Strategie auch für die Zustände $s_1, s_2, s_3, s_5, s_6, s_7$, wenn er der heuristischen Information mit $k > 0$ folgt. Diese Tatsache resultiert im schnelleren Lernen, wenn die Parameter richtig eingestellt sind.
Im Szenario mit Sackgassen steht die heuristische Funktion auch in Konflikt mit der optimalen Lösung, siehe Abbildung 4.3(a). Die Zustände s_2, s_8 bzw. s_5, s_{11} repräsentieren eine Art Sackgasse. In dieser Umgebung existieren mehrere Zustände, in denen die Aussagen der heuristischen Funktion nicht mit Aussagen der optimalen Strategie übereinstimmen. Im Fall $k > 0$ ist solch ein Zustand s_{17}, für $k < 0$ sind solche Zustände s_2, s_3, s_5. Wenn sich der Agent im Zustand s_8 oder s_{14} befindet, besagt die heuristische Funktion, dass die beste Aktion eine Bewegung nach oben zum Zustand s_2 ist, was zu einem Konflikt mit der optimalen Strategie führt, vergleiche Abbildung 4.3(b). Für die Zustände s_3 und s_5 gilt dasselbe Prinzip.

4.2.3 Zyklen, Zykluskandidaten, zulässiges Intervall

Die Auswirkung der heuristischen Funktion auf die Konvergenzeigenschaften des erweiterten Lernalgorithmus zur gesuchten Strategie ist eine wichtige Fragestellung. Am Beispiel der Umgebung mit Weggabelung in der Gitterwelt wurde gezeigt, dass die angewendete Heuristik in Konflikt zur optimalen Strategie stehen kann. Bei falscher Parametrisierung der heuristischen Funktion kann dieser Konflikt sogar zur Divergenz des Lernalgorithmus führen. An dieser Stelle werden für die Konvergenz der erweiterten Lernalgorithmen wichtige Begriffe wie Zyklus, Terminalzustand und Zykluskandidat eingeführt und am Beispiel des Gitterweltproblems erläutert.

Definition 4.2.1 (Zyklus). *Ein Zyklus ist eine Kette von unendlich vielen, sich wiederholenden Schritten. Seine Periode enthält T Schritte. Die Kette kehrt nach Schritt T zum Anfang des Zyklus zurück, also gilt:* $(s_t, s_{t+1}, \ldots, s_{t+T-1}, s_{t+T} = s_t, \ldots)$.
Ein Zyklus, der aus nur einem Element besteht $(s_T, s_T = s_{T+1}, \ldots)$*, ist ein vom Ziel unterscheidbarer Terminalzustand.*

Definition 4.2.2 (Zykluskandidat). *Ein Zykluskandidat ist der Zustand, in dem die heuristische Funktion in Konflikt zur optimalen Strategie steht. In einem Zykluskandidat kann bei Anwendung des erweiterten RL-Algorithmus mit einer falsch parametrisierten heuristischen Funktion ein Zyklus entstehen. In den theoretischen Überlegungen wird nur der Zykluskandidat betrachtet, der aus einem minimalen Zyklus besteht. Der minimale Zyklus kann aus einem sich wiederholenden Zustand oder aus zwei sich wiederholenden Zuständen bestehen.*

Der erweiterte Lernalgorithmus divergiert, wenn er in einer endlichen Anzahl der Lernschritte nicht zur gesuchten Strategie konvergiert. Die gesuchte Strategie steuert den Agenten zum Zielzustand. Dieser Fall tritt dann ein, wenn Zustände existieren, von denen aus der Agent das Ziel nie erreicht. Dies kann ein sich vom Zielzustand unterscheidbarer Terminalzustand oder ganze sich wiederholende Ketten von Zuständen (Zyklen) sein, die nach Meinung des Agenten eine größere Belohnung einbringen als das Erreichen des Ziels.

Wenn Zyklen bei der Anwendung des erweiterten Lernalgorithmus entstehen, kann dies durch eine zufällige Aktionswahl, die in der ϵ-gierigen Strategie mit der Wahrscheinlichkeit ϵ auftritt, teilweise vermieden werden. Wenn alle benachbarten Zustände zu einem Zyklus führen, wird es mit der ϵ-gierigen Strategie sehr schwer den Agenten aus dem Zyklus herauszubringen.
Zur Verdeutlichung des Begriffs Zykluskandidat soll ein Beispiel für mögliche Zyklen in der Umgebung mit Weggabelung, siehe 4.2(a), dienen.
Für den Fall $k < 0$ besagt die verfolgte Heuristik, dass das Erreichen des Zustandes s_1 die beste Strategie ist. Die eigentlich optimale Strategie des Agenten besagt, dass der Agent auf dem schnellsten Weg den Zustand s_{11} erreichen soll. Der Agent soll den negativen Einfluss der heuristischen Funktion während des Lernprozesses zuerst bewältigen. Bildlich kann die heuristische Funktion als ein Hügel betrachtet werden, mit der Bergspitze auf dem vom Ziel am weitesten entfernten Zustand. Der Agent muss erst einmal bergauf steigen und den weitesten Zustand s_{11} in der Umgebung erreichen. Ab diesem Punkt stimmt die Aussage der heuristischen Funktion mit der gesuchten Strategie, die den Agenten zum Ziel steuert, überein, und die heuristische Funktion beeinflusst die Suche nach der Lösung positiv. Der Aufstieg verursacht erst einmal Kosten. In manchen Fällen ist es aus Sicht des Agenten besser sich neben dem Berg aufzuhalten, statt den hohen Berg zu erklimmen, wenn das Ziel nicht einmal in Sicht ist. Aus dieser Überlegung heraus kann die Vermutung über die Abhängigkeit des gültigen Parameterwertes für k von der Sichtweite γ des Agenten, auch als Diskontierungsfaktor bezeichnet, aufgestellt werden. Somit kann an der Stelle s_1 mit einem nicht korrekt eingestellten Parameterpaar für (k, γ) der Zyklus $(s_1, s_5, s_1, s_5, \cdot)$ für das Belohnungsmodell mit Sonderbestrafung, oder der Zyklus $(s_1, s_1, s_1, s_1, \cdot)$ für das Standard-Belohnungsmodell entstehen. Wenn der Agent in diesen Zyklus gerät, wird er in diesem Zustand stecken bleiben und nie herauskommen. In diesem Zustand oder einer Menge von Zuständen steht dann die heuristische Funktion in Konflikt zur eigentlich optimalen Strategie, die den Agenten zum Ziel führen soll. Und diese Zustände s_1, s_5 werden dann als Zykluskandidaten bezeichnet.
Für $k > 0$ kann im Zustand s_{11} der Zyklus $(s_{11}, s_7, s_{11}, s_7, \ldots)$ bzw. $(s_{11}, s_{10}, s_{11}, s_{10}, \ldots)$ entstehen, wenn die Kollisionen mit der Wand bestraft werden, oder auch der Zyklus $(s_{11}, s_{11}, s_{11}, s_{11}, \ldots)$ entstehen, wenn die Parameterkombination (k, γ) nicht richtig eingestellt ist, also die Belohnung für das Erreichen des Zustandes s_{11} zu groß ist. Für $k > 0$ wird der Zustand s_{11} als Zykluskandidat bezeichnet.
Die Bedingungen, unter denen bei Anwendung des erweiterten Lernalgorithmus keine Zyklen entstehen, werden eingeführt. Die Tatsache, dass durch eine Garantie für das Nicht-Entstehen von Zyklen der erweiterte Algorithmus nach einer endlichen Anzahl an Lernschritten zur optimalen Lösung konvergiert, wird nur angenommen. Es gilt die wichtige Annahme, dass der Standard-SARSA-Algorithmus nach einer endlichen Anzahl an Lernschritten eine gute Strategie erlernt, die den Agenten zum Zielzustand steuert.
Die Wahl des heuristischen Einflussparameters k wird in Abhängigkeit vom Parameter γ des Standard-SARSA-Algorithmus bestimmt.

Definition 4.2.3 (Zulässiges Intervall). *Ein Intervall für den Einflussparameter k wird hier als zulässig bezeichnet, wenn für alle k aus diesem Intervall die Konvergenz des erweiterten Lernalgorithmus zur Strategie, die den Agenten zum Ziel steuert, garantiert werden kann. Dabei gilt die Annahme, dass der nicht erweiterte Lernalgorithmus zu einer Strategie konvergiert, die den Agenten zum Zielzustand navigiert.*

In späteren Kapiteln wird gezeigt, dass für die meisten Szenarien gute Ergebnisse mit folgender Parametrisierung $\gamma \in [0.95, 1)$ und $k \in [-0.5, -0.1]$ und bei Anwendung des Standard-Belohnungsmodells erzielt werden.

4.3 Bestimmung des zulässigen Intervalls für die ϵ^*-gierige Strategie

In diesem Kapitel werden die ersten Überlegungen über die Bestimmung des zulässigen Intervalls für die Garantie der Konvergenz zur gesuchten Lösung für den Standard-SARSA-Algorithmus bei Anwendung der ϵ^*-gierigen Strategie eingeführt. Diese Überlegungen und die vorgenommenen Abschätzungen werden zuerst für das Gitterweltproblem mit Standard-Belohnungsmodell durchgeführt.
Die ϵ^*-gierige Strategie besteht aus zwei Bestandteilen, siehe Gleichung (4.1). Der erste Teil ist die aktuelle Repräsentation der Q-Funktion. Der zweite Teil ist die Bewertung der zukünftigen Zustände $s' := \tau(s, a)$ durch die heuristische Funktion, also wie gut sich die Aktion a aus Sicht der heuristischen Funktion auswirkt.

4.3.1 Zulässiges Intervall für den heuristischen Einflussparameter k

Um garantieren zu können, dass die Konvergenz des Lernprozesses zu einer guten Lösung[24] durch das Hinzufügen einer heuristischen Funktion in die Bildung der Strategie nicht verletzt wird, dürfen keine Zyklen in den Zuständen des Zustandsraumes entstehen, die als Zykluskandidaten erkannt worden sind. Die Problematik der Entstehung der Zyklen wurde im vorherigen Kapitel thematisiert. Zyklen können vermieden werden, indem der heuristische Einflussparameter k im vordefinierten zulässigen Intervall liegt, siehe Ungleichung (4.7). Die Ermittlung des zulässigen Intervalls wird im Beweis zum Satz 4.3.1 aufgedeckt. Der Beweis wird für das Gitterweltproblem und für die als normierte, parametrisierte Abstandsfunktion zum Ziel definierte heuristische Funktion durchgeführt, kann aber mit einem entsprechenden Diskretisierungsschritt auch für das Navigationsproblem und das MCP angewendet werden. Denn die angewendete heuristische Funktion realisiert in allen drei Problemarten die gleiche Heuristik ”je näher der Zustand am Zielzustand liegt, desto besser”. Die Lerngeschwindigkeit, sowie die Qualität der ermittelten Lösungen im Vergleich zum Standard-SARSA-Algorithmus wird in späteren Unterkapiteln in unterschiedlichen Problemstellungen empirisch untersucht.

Satz 4.3.1. *Für die Garantie für das Nicht-Entstehen von Zyklen an den Zykluskandidaten während der Anwendung der erweiterten ϵ^*-gierigen Strategie im SARSA-Algorithmus soll der Parameter k in Abhängigkeit vom im SARSA-Algorithmus angewendeten Parameter $0 < \gamma \leq 1$ und für die im Vorfeld ermittelten Parameter $T_>, T_<$ bestimmt werden, so dass die Ungleichung (4.7) gilt. Das mit der Ungleichung bestimmte Intervall wird auch als zulässiges Intervall für den heuristischen Einflussparameter k bezeichnet. Für den Zykluskandidaten $s_<$ mit $k < 0$ wird der Parameter $T_<$ als maximale Anzahl der benötigten Schritte bei Anwendung der bekannten Strategie vom Zustand $s_<$ aus bis zum Erreichen des Zielzustands berechnet. Für den Zykluskandidaten $s_>$ mit $k > 0$ wird der Parameter $T_>$ als die maximale Anzahl der bis zum Erreichen des Zielzustands benötigten Schritte unter der Anwendung der bekannten Strategie vom Zustand $s_>$ berechnet.*

$$-\gamma^{T_<+1} \cdot \frac{Norm}{2} \leq k \leq \gamma^{T_>+1} \cdot \frac{Norm}{2} \tag{4.7}$$

Bemerkung 4.3.1. *Im Gitterweltproblem für das Szenario mit Weggabelung wäre für $k < 0$ dann ein solcher Zykluskandidat $s_< := s_1$. Der Parameter $T_<$ wird auf den Wert 9 gesetzt. Für den Wert $k > 0$ wird der Zykluskandidat $s_> = s_{11}$. Der Wert von $T_>$ wird zu 5 berechnet, $Norm = \frac{1}{13}$.*

[24] Als gute Lösung wird eine Strategie bezeichnet, die den Agenten schnell in einer endlichen Anzahl an Lernschritten zum Ziel führt.

Beweis. Sei $Q^*(s,a)$ die mit dem Standard-SARSA-Algorithmus erlernte Bewertungsfunktion. Optimal bedeutet, dass das Ziel vom Zustand s aus in minimaler Anzahl an Schritten T_s erreicht wird, wenn den Empfehlungen der Q^*-Funktion gefolgt wird. Wenn eine Aktion gewählt wird, die nicht mit den Empfehlungen der optimalen Strategie $\pi^*(s)$, die aus der optimalen Q^*-Funktion abgeleitet wird, übereinstimmt, soll der Agent mindestens einen Schritt mehr ausführen als bei Befolgung der Empfehlungen der optimalen Strategie. Diese Tatsache wird in Ungleichung (4.8) unter der Annahme des Standard-Belohnungsmodells formalisiert. Es ist zu bemerken, dass $\pi^*(s) := \{a_1^*, \dots a_M^*\}$ mehrere Aktionen vom Zustand s aus als Empfehlung enthalten kann, siehe z. B. Zustand s_1 in Abbildung 4.2(c).

$$Q^*(s,a^*) - Q^*(s,a) \geq -1 \cdot \sum_{t=0}^{T_s} \gamma^t - (-1 \cdot \sum_{t=0}^{T_s+1} \gamma^t) =: \gamma^{T_s+1} \tag{4.8}$$

wobei $a \notin \pi^*(s)$ und $a^* \in \pi^*(s)$.
Wenn der Umweg, also die Differenz zwischen dem tatsächlichen Weg und dem optimalen Weg des Agenten mehr als einen Schritt beträgt, wird die Ungleichung wie folgt erweitert

$$\underbrace{Q^*(s,a^*) - Q^*(s,a) \underbrace{\geq \gamma^{T_s+1}}_{I} \geq \gamma^{T_s+P}}_{II}.$$

Dann wird die Geltung von I automatisch die Geltung von II hervorrufen.
Für die erweiterte Methode soll die folgende Beziehung (4.9) erfüllt sein, um die Verletzung der Optimalität zu verhindern:

$$Q^*(s,a^*) + h(\underbrace{\tau(s,a^*)}_{s'}) \geq Q^*(s,a) + h(\underbrace{\tau(s,a)}_{s'}) \tag{4.9}$$

wobei $a^* \in \pi^*(s)$ und $a \notin \pi^*(s)$. Hier werden nur die Zustände betrachtet, in denen die Empfehlung der heuristischen Funktion nicht mit der Empfehlung der Strategie π^* übereinstimmt, also der Zykluskandidat s mit $h(\tau(s,a^*)) < h(\tau(s,a))$. Je nach Vorzeichen des Einflussparameters k wird dieser Widerspruch an unterschiedlichen Zuständen vorkommen, da $0 < h(\tau(s,a)) - h(\tau(s,a^*)) \overset{(4.4)}{:=} k \cdot (\widetilde{h}(\tau(s,a)) - \widetilde{h}(\tau(s,a^*)))$ gilt.
Für $k > 0$ wird es an den Zuständen $s^1_>, s^2_>$, wo $(\widetilde{h}(\underbrace{\tau(s_>,a)}_{=s^1_>}) - \widetilde{h}(\underbrace{\tau(s_>,a^*)}_{=s^2_>})) > 0$, gefährlich, und die Gültigkeit der Ungleichung (4.9) kann verletzt werden.
Für $k < 0$ sind die Zustände $s^1_<, s^2_<$, wo $(\widetilde{h}(\underbrace{\tau(s_<,a)}_{=s^1_<}) - \widetilde{h}(\underbrace{\tau(s_<,a^*)}_{=s^2_<})) < 0$ gilt, gefährlich und können die Gültigkeit der Ungleichung (4.9) verletzen. Sonst gilt die Ungleichung (4.9) und die Optimalität wird nicht verletzt. Folglich soll gelten:

$$Q(s,a^*) - Q(s,a) \overset{(4.8)}{\geq} \gamma^{T_s+1} \overset{z.Z.}{\geq} h(\tau(s,a)) - h(\tau(s,a^*)) =: k \cdot (\widetilde{h}(\tau(s,a)) - \widetilde{h}(\tau(s,a^*))) \tag{4.10}$$

Wenn mehrere Zykluskandidaten im Szenario auftreten können, kann nur ein "extremer" Zykluskandidat behandelt werden, so dass die durchgeführten Überlegungen auch für andere Zykluskandidaten gelten.

Die Ungleichung (4.10) wird wie folgt in zwei Ungleichungen umformuliert:

$$\begin{aligned} Q(s_>, a^*) - Q(s_>, a) &\overset{(4.8)}{\geq} \gamma^{T_>+1} \overset{z.Z.}{\geq} h(\tau(s_>, a)) - h(\tau(s_>, a^*)) \\ &=: \underbrace{k}_{>0} \cdot (\underbrace{\widetilde{h}(\tau(s_>, a)) - \widetilde{h}(\tau(s_>, a^*))}_{>0}) \end{aligned} \tag{4.11}$$

$$\begin{aligned} Q(s_<, a^*) - Q(s_<, a) &\overset{(4.8)}{\geq} \gamma^{T_<+1} \overset{z.Z.}{\geq} h(\tau(s_<, a)) - h(\tau(s_<, a^*)) \\ &=: \underbrace{k}_{<0} \cdot \underbrace{(\widetilde{h}(\tau(s_<, a)) - \widetilde{h}(\tau(s_<, a^*)))}_{<0} \end{aligned} \tag{4.12}$$

Mit der Definition der heuristischen Funktion wie in Gleichung (4.4) definiert für einen metrischen Raum im Navigationsproblem lässt sich die entstandene Differenz wie folgt abschätzen:

$$\max_a |\widetilde{h}(\underbrace{\tau(s, a)}_{s''}) - \widetilde{h}(\underbrace{\tau(s, a^*)}_{s'}| \leq \frac{2}{Norm} \tag{4.13}$$

Wenn zwei unterschiedliche Aktionen von einem Zustand aus gemacht werden, und sich die Aktionen unterscheiden, werden zwei ”inverse” Aktionen benötigt, um von einem Zustand, z. B. s'', zum anderen Zustand, z. B. s', zu gelangen. Wenn die Aktionen mit räumlichen Veränderungen verbunden sind, kann der Abstand zwischen zwei Zuständen als Wert $\frac{2}{Norm}$ abgeschätzt werden. Aus dieser Überlegung ist diese Abschätzung entstanden. Diese Abschätzung hängt vom Aufbau des Zustandsraums und der Translationsfunktion ab.
Im Folgenden wird die durchgeführte Abschätzung (4.13) in der Gitterwelt eingeführt, um die aufgestellte Abschätzung besser nachvollziehen zu können. Für den Zustand s_2 im Beispiel 4.2(a) ist der Vorschlag der optimalen Strategie $a^* = \rightarrow$. Die heuristische Funktion liefert $a = \leftarrow$, so dass die maximal mögliche Distanz zwischen den Zuständen $d(s', s'') \overset{4.6}{\leq} \frac{2}{Norm}$ ist.
Durch Einsetzen der vorgenommenen Approximationen (4.11), (4.12) und (4.13) in die Ungleichung (4.10) kann das zulässige Intervall für den Parameter k wie folgt formuliert werden:

$$-\gamma^{T_<+1} \leq \frac{2 \cdot k}{Norm} \leq \gamma^{T_>+1}$$

Dies ist die aufgestellte Bedingung für die Definition der heuristischen Funktion, siehe Ungleichung (4.7). □

Bemerkung 4.3.2. *Wenn das Navigationsproblem für den stetigen Zustandsraum formuliert ist, gilt die oben aufgestellte Abschätzung (4.13) auch im Fall einer richtig gewählten Auflösung der Codierung. Wenn zwei Zustände in einem Teil durch Ausführen einer Aktion nicht unterschieden werden, ist der Zerlegungsschritt zu grob gewählt. Die vorgenommene Konfiguration des Standard-SARSA-Algorithmus ist nicht optimal. Wenn der Agent direkt zwei Teile durch Ausführung einer Aktion überspringt, ist die gewählte Diskretisierung dann zu fein.*

Bemerkung 4.3.3. *Für den Fall, dass das Belohnungsmodell um eine Sonderbelohnung erweitert wird, können folgende Überlegungen gemacht werden. Das Belohnungsmodell mit Sonderbelohnung ist wie folgt definiert. Für jeden Zeitschritt wird der Agent mit einem konstanten Wert von -1 bestraft. Zusätzlich erhält der Agent für das Erreichen des Ziels eine Sonderbelohnung von $+100$.*
Wenn der Agent nach dem Erreichen des Ziels eine Belohnung erhält, wird der Wert auf der rechten Seite

der Ungleichung größer. Wenn z. B. die positive Belohnung $+100$ *ist, sieht die Ungleichung (4.8) wie folgt aus:* $Q^*(s,a^*) - Q^*(s,a) \geq -1 \cdot \sum_{t=0}^{T-1} \gamma^t + 100 \cdot \gamma^T + \sum_{t=0}^{T} \gamma^t - 100 \cdot \gamma^{T+1} =: \gamma^T + \gamma^T \cdot 100 \cdot (1-\gamma)$. *Somit kann die Ungleichung (4.7) für das Belohnungsmodell mit Sonderbelohnung wie folgt abgeschwächt werden:*

$$-1(\gamma^{T<} + \gamma^{T<} \cdot 100 \cdot (1-\gamma)) \cdot \frac{Norm}{2} \leq k \leq \underbrace{(\gamma^{T>} + \gamma^{T>} \cdot 100 \cdot (1-\gamma)) \cdot \frac{Norm}{2}}_{\textit{Belohnungsmodell mit Sonderbelohnung}}$$

4.3.2 Manuell bestimmte Grenzwerte für das zulässige Intervall

Zur Bestätigung der theoretischen Ergebnisse werden die Grenzwerte für das zulässige Intervall des heuristischen Einflussparameters k für beide vorgeschlagenen heuristisch erweiterten Algorithmen in Abhängigkeit von den unterschiedlichen Ausprägungen des Diskontierungsfaktors durch eine Gittersuche bestimmt. Bei der Analyse des theoretisch hergeleiteten, heuristischen Intervalls wird angenommen, dass eine bekannte, optimale Strategie im untersuchten Gitterweltproblem vorhanden ist, die den Agenten von jedem Zykluskandidaten zum Ziel navigiert.
In einem Lernschritt werden die Werte für die Q-Funktion für alle Zustände nacheinander mithilfe des erweiterten Lernschritts für den SARSA*-Algorithmus in Gleichung (4.2) bzw. des Lernschritts für das Lernen mit dem SARSA-Algorithmus in Gleichung (3.10) angepasst. Wenn die Anpassung der Q-Funktion für einen Lernschritt abgeschlossen ist, wird der Lernschritt durch Bildung der aktuellen Strategie für die aktuelle Q-Funktion mithilfe der gierigen Wahl bzw. mithilfe der heuristisch erweiterten gierigen Wahl abgeschlossen, siehe Gleichung (4.1).
Die Lernschritte werden solange durchgeführt, bis die gebildete Strategie sich nicht mehr ändert. Die Anzahl der bis zur Konvergenz benötigten Schritte wird festgehalten. Für die Prüfung der Stabilität werden ein paar zusätzliche Lernschritte durchgeführt. Diese Schritte werden nicht in den Ergebnissen berücksichtigt. Eine solche Vorgehensweise ist einem Beispiel der Gitterwelt von Sutton entnommen [Sutton & Barto, 1998, Kap. 4.2, s. 94].
Ein *kritischer Wert* für den heuristischen Einflussparameter k ist wie folgt definiert: Für den kritischen Wert $k(\gamma) = c$ findet der heuristisch erweiterte Algorithmus nach einer endlichen Anzahl an Schritten eine Lösung, die mit der vordefinierten optimalen Strategie übereinstimmt. Wenn der heuristische Einflussparameter k je nach Vorzeichen von k_{Grenze} den Wert $(k_{Grenze} + \delta)$ bzw. $(k_{Grenze} - \delta)$ annimmt[25], divergiert der heuristisch erweiterte Algorithmus, d. h. es entsteht mindestens ein Zyklus im Zykluskandidaten in der erlernten Strategie.
Die Gittersuche wird in einem zweidimensionalen Parameterraum durchgeführt. Dabei wird für den festgelegten Parameter γ der kritische Wert für den heuristischen Einflussparameter bestimmt.
Für jedes $\gamma = 0.5, 0.51, \ldots, 1.0$ wird ein kritischer Wert des heuristischen Einflusses k_{Grenze} für beide Varianten des heuristisch erweiterten SARSA-Algorithmus gesucht. Diese Vorgehensweise wird durch eine Gittersuche je nach Vorzeichen für k realisiert. Angefangen bei Null wird je nach Vorzeichen des untersuchten Grenzwertes ein Delta-Wert addiert bzw. subtrahiert, so dass der Parameter k folgende Werte annimmt: $0, (+/-\delta), (+/-2 \cdot \delta)$. Dieser Prozess wird solange durchgeführt, bis der kritische Wert für den Parameter $k = k_{Grenze}$ erreicht ist.
Mit der beschriebenen Vorgehensweise wird das zulässige Intervall für den heuristischen Einflussparameter k für unterschiedliche Ausprägungen des Diskontierungsfaktors γ manuell bestimmt.

[25] wobei δ auf den Wert 0.01 gesetzt wird

4.3.3 Zulässiges Intervall für das Gitterweltproblem

Das theoretisch ermittelte, zulässige Intervall des heuristischen Einflussparameters k, siehe Ungleichung (4.7), wird mit dem manuell ermittelten, zulässigen Intervall für unterschiedliche Parameter γ verglichen, um theoretische Aussagen in Bezug auf die Anwendung der ϵ^*-gierigen Strategie zu bestätigen. Diese Analyse wird für das Gitterweltproblem mit zwei Szenarien, Umgebung mit Weggabelung und Umgebung mit Sackgassen, durchgeführt.
Die Umgebung mit Weggabelung hat eine Größe von 3×4, der Zielzustand ist s_0, $Norm := \max_{s_i} ||s_0 - s_i|| := \sqrt{3^2+2^2} = \sqrt{13}$, siehe Abbildung 4.2(a). Für diese Umgebung werden die theoretischen Grenzwerte für das zulässige Intervall des Parameters $k(\gamma)$ wie folgt bestimmt.
Für $k < 0$ ist der Zykluskandidat der Zustand $s_< := s_1$, da dieser "extreme" Zustand am weitesten vom Zielzustand entfernt liegt, und die heuristische Funktion für diesen Zustand den besten Wert liefert, außer für das Erreichen des Ziels. Die minimale Anzahl der Schritte vom Zykluskandidaten s_1 bis zum Zielzustand s_0 beträgt $T_< = 9$ Schritte, da der optimale Weg des Agenten vom Zustand des Zykluskandidaten bis zum Zielzustand die folgenden 9 Schritte benötigt:

$$(s_1,\downarrow),(s_5,\rightarrow),(s_6,\rightarrow),(s_7,\downarrow),(s_{11},\leftarrow),(s_{10},\leftarrow),(s_9,\leftarrow),(s_8,\uparrow),(s_4,\uparrow).$$

Für $k > 0$ ist der Zykluskandidat der Zustand $s_> := s_{11}$, da dieser Zustand am weitesten vom Zielzustand entfernt liegt. Vom Zykluskandidaten s_{11} werden 5 Schritte bis zum Erreichen des Ziels benötigt, wenn die optimale Strategie verfolgt wird, so dass $T_> = 5$ ist.
Die Abschätzung des zulässigen Intervalls für den heuristischen Einflussparameter k in der ϵ^*-gierigen Strategie für die Umgebung mit Weggabelung, vergleiche Ungleichung (4.7), sieht wie folgt aus:

$$\underbrace{-\gamma^{10}\cdot\frac{\sqrt{13}}{2}}_{\text{Abschätzung für die untere Grenze}} \leq k \leq \underbrace{\gamma^{6}\cdot\frac{\sqrt{13}}{2}}_{\text{Abschätzung für die obere Grenze}} \tag{4.14}$$

In Abbildung 4.5 sind die manuell und theoretisch bestimmten Grenzwerte für das zulässige Intervall des Parameters k in Abhängigkeit vom Diskontierungsfaktor γ dargestellt. Dabei kann eine gute Approximation der manuell bestimmten Grenzwerte durch die theoretisch bestimmten Werte beobachtet werden. Die beiden Kurven nähern sich mit absteigenden Werten von γ an.
Um die bestimmten Ergebnisse aus der Umgebung mit Weggabelung zu bestätigen, werden die Grenzwerte für das zulässige Intervall des Parameters k manuell und theoretisch für die Umgebung mit Sackgassen bestimmt, siehe Abbildung 4.3(a). Die Umgebung mit Sackgassen hat die Größe 6×3, der Zielzustand ist der Zustand s_0, $Norm = \sqrt{29} = \sqrt{5\cdot5+2\cdot2}$.
Für $k > 0$ ist der Zykluskandidat der Zustand $s_> := s_{17}$. Die Anzahl der minimalen Schritte bis zum Ziel beträgt $T_> = 9$ Schritte. Vom Zustand s_{17} werden die folgenden neun Schritte bis zum Ziel benötigt:

$$(s_{17},\leftarrow),(s_{16},\uparrow),(s_{10},\leftarrow),(s_9,\downarrow),(s_{15},\leftarrow),(s_{14},\leftarrow),(s_{13},\uparrow),(s_7,\uparrow),(s_1,\uparrow).$$

Für $k < 0$ ist der "extreme" Zykluskandidat der Zustand $s_< := s_5$. In diesem Zustand stehen die Aussagen der heuristischen Funktion und der optimalen Strategie in Konflikt zueinander. Die minimale Anzahl der bis zum Ziel benötigten Schritte beginnend bei Zustand s_5 ist $T_< = 11$, wobei der optimale Weg den Teilweg vom Zustand s_{17}, siehe die neun oben genannten Schritte, beinhaltet. Vom Zustand s_5 bis zum Zustand s_{17} werden weitere zwei Schritte benötigt. Insgesamt liegt der Wert bei $T_< = 11$. Der Zustand s_2 ist auch ein möglicher Zykluskandidat, die minimale Anzahl der bis zum Ziel benötigten Schritte vom

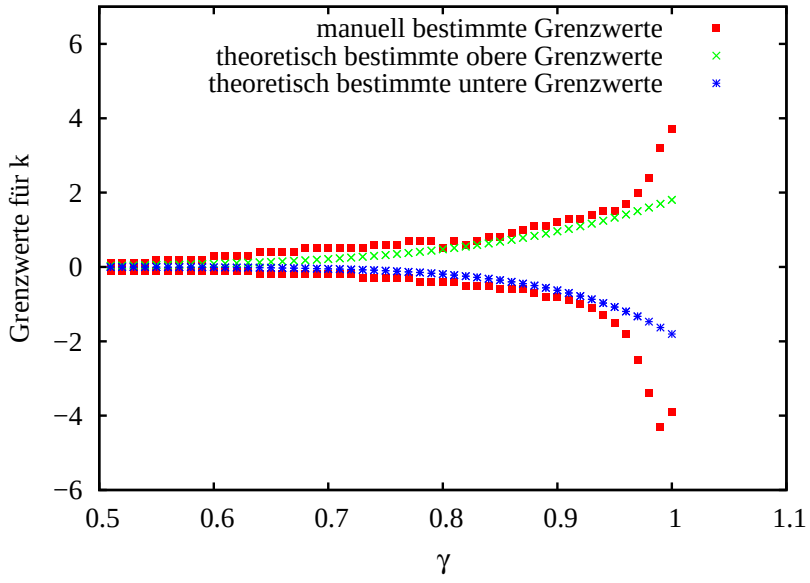


Abbildung 4.4: Abhängigkeit der Grenzwerte des zulässigen Intervalls für den Parameter k vom Diskontierungsfaktor γ für die Umgebung mit Weggabelung in der Gitterwelt, manuell bzw. theoretisch berechnet. $T_> = 5$ ist die Anzahl der Schritte vom Zykluskandidat $s_> := s_{11}$ bis zum Zielzustand s_0. $T_< = 9$ ist die Anzahl der Schritte vom Zykluskandidaten $s_< := s_1$ bis zum Zielzustand.

Zustand s_2 ist $\widetilde{T}_< = 6$. Dieser Wert $\widetilde{T}_< = 6$ ist kleiner als $T_< = 11$. Wenn das Problem der Entstehung der Zyklen im Zustand s_5 durch richtige Parametrisierung des heuristischen Einflussparameters k behoben wird, dann wird auch kein Zyklus im Zustand s_2 entstehen ($-\gamma^6 < -\gamma^{11}$). Eine Abschätzung für die Grenzwerte sieht wie folgt aus:

$$-\gamma^{11} \cdot \frac{\sqrt{29}}{2} \leq k \leq \gamma^{10} \cdot \frac{\sqrt{29}}{2}$$

Die Grenzwerte werden mit der oben beschriebenen Vorgehensweise manuell bestimmt. Die theoretisch und manuell bestimmten Grenzwerte für das zulässige Intervall sind in Abhängigkeit vom Wert für γ in Abbildung 4.5 abgebildet. Die Übereinstimmung der theoretischen Aussagen mit den praktisch erzielten Aussagen wird in der Umgebung mit Sackgassen bestätigt. Die beiden Kurven liegen nah beieinander.
Für beide Szenarien des Gitterweltproblems werden die in diesem Kapitel hergeleiteten theoretischen Ergebnisse durch praktische Ergebnisse bestätigt. Bei der Bestimmung des zulässigen Intervalls für den Parameter k kann eine gute Approximation der manuell erzielten Grenzwerte durch theoretisch bestimmte Grenzwerte beobachtet werden.

4.4 Bestimmung des zulässigen Intervalls für den SARSA*-Algorithmus

An dieser Stelle wird die Bestimmung des zulässigen Intervalls für den heuristischen Einflussparameter k im SARSA*-Algorithmus, siehe Lernregel aus Gleichung (4.2), durchgeführt. Auch hier stehen die Überlegungen und Abschätzungen in einer starken Verbindung mit dem Navigationsproblem im Gitterweltproblem. Es werden Bedingungen aufgestellt, in denen der erweiterte SARSA*-Algorithmus unab-

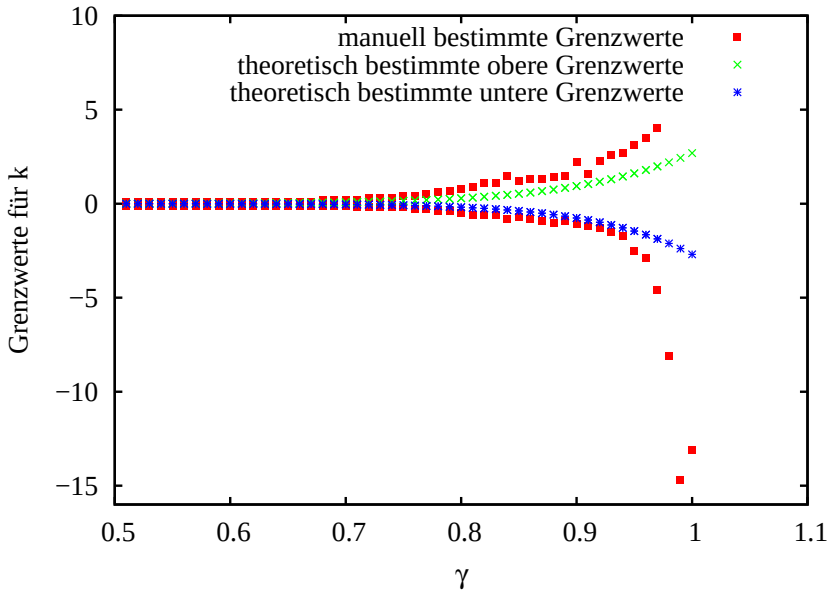


Abbildung 4.5: Abhängigkeit der Grenzwerte des zulässigen Intervalls für den Parameter k vom Diskontierungsfaktor γ für die Umgebung mit Sackgassen in der Gitterwelt, manuell bzw. theoretisch berechnet. $T_> = 9$ ist die Anzahl der Schritte vom Zykluskandidaten $s_> := s_{17}$ bis zum Zielzustand s_0. $T_< = 11$ ist die Anzahl der Schritte vom Zykluskandidaten $s_< := s_5$ bis zum Zielzustand.

hängig vom Startzustand nie in einem Zyklus stecken bleibt, wenn der Standard-SARSA-Algorithmus zur gesuchten Lösung konvergiert. Der weitere Schritt, der Schluss aus dem Nicht-Entstehen von Zyklen auf die Konvergenz des SARSA*-Algorithmus zur optimalen Lösung, wird in der vorliegenden Arbeit nur intuitiv vermutet. Die erzielten theoretischen Abschätzungen für den heuristischen Einflussparameter im Vergleich zum mit der Gittersuche manuell bestimmten zulässigen Intervall im Gitterweltproblem, sowie die beobachtete Steigerung der Lerngeschwindigkeit für unterschiedliche Problemstellungen bei Anwendung des SARSA*-Algorithmus weisen auf die Richtigkeit der aufgestellten Vermutung hin.

4.4.1 Grundidee für den Beweis

Wie im vorherigen Kapitel werden hier die Zykluskandidaten untersucht, bei denen die Aussagen der heuristischen Funktion in Konflikt zur Aussage der bekannten Strategie stehen. Die Auswirkung der heuristischen Funktion auf den Lernprozess mit dem SARSA*-Algorithmus ist komplexer als die Anwendung der heuristischen Funktion für die Bildung der Strategie.
Für die Garantie, dass das Bleiben in einem Zyklus für den Agenten ”schlechter” ist als eine Bewegung in Richtung des Ziels, muss die folgende Ungleichung (4.15) gelten. In diesem Fall ist es für den Agenten vorteilhafter zum Ziel zu steigen als im Zyklus zu bleiben. Wenn keine Zyklen im Zykluskandidaten s_z entstehen[26], wird angenommen, dass der Lernprozess die gesuchte Lösung nach einer endlichen Anzahl an Schritten erlernen wird.

$$Q^{\pi^*}(s_z, \pi^*(s_z)) \geq Q^{Zyklus}(s_z, \pi^{Zyklus}(s_z)) \tag{4.15}$$

[26] s_z ist die Bezeichnung für einen Zykluskandidaten, unabhängig vom Vorzeichen des heuristischen Einflussparameters k.

Also soll der gebildete Wert der Gesamtbelohnung in einem Zykluskandidaten für den Zyklus (unendliche Summe $Q^{Zyklus}(s_z, \pi^{Zyklus}(s_z))$), bei dem die Strategie des Agenten aus Aktionen besteht, die den Agenten in eine wiederholende Reihenfolge der Zustände führt, keine besseren Werte als die Gesamtbelohnung ($Q^{\pi^*}(s_z, \pi^*(s_z))$) für eine Strategie, die den Agenten zum Ziel steuert, aufweisen. Der Wert $Q^{\pi^*}(s_z, \pi^*(s_z))$ besteht für den SARSA*-Algorithmus aus den elementaren Belohnungen des Belohnungsmodells und den Werten der heuristischen Funktion. Das zulässige Intervall für den heuristischen Einflussparameter k soll so definiert sein, dass die aufgestellte Ungleichung (4.15) gilt.

4.4.2 Hilfsmittel

Bemerkung 4.4.1. *Für $0 \leq \gamma < 1$ gilt:*

$$\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

Bemerkung 4.4.2. *Für die Subtraktion einer endlichen Reihe von einer unendlichen Reihe gilt:*

$$\sum_{t=0}^{\infty} \gamma^t - \sum_{t=0}^{T} \gamma^t = \sum_{t=T+1}^{\infty} \gamma^t = \sum_{t=0}^{\infty} \gamma^{t+T+1} = \gamma^{T+1} \cdot \sum_{t=0}^{\infty} \gamma^t = \frac{\gamma^{T+1}}{1-\gamma}$$

Bemerkung 4.4.3. *Für $a+b < \infty$ gilt $\frac{a}{a+b} = 1 - \frac{b}{a+b}$. Sei $a := \sum_{t=0}^{T} \gamma^t$, $b := \gamma^{T+1} \sum_{t=0}^{\infty} \gamma^t$, dann gilt zusammen mit Bemerkung 4.4.2 und der Abschätzung $a + b = \sum_{t=0}^{\infty} \gamma^t < \infty$ für $0 < \gamma < 1$:*

$$\frac{\sum_{t=0}^{T} \gamma^t}{\sum_{t=0}^{\infty} \gamma^t} = 1 - \gamma^{T+1}$$

4.4.3 Zulässiges Intervall für den heuristischen Einflussparameter k

Für jeden Zykluskandidaten $s_z \in S$ sollen die Parameter (γ, k) so bestimmt werden, dass die Ungleichung (4.15) gilt und in keinem der Zykluskandidaten verletzt wird.
Für einen aus einem Schritt bestehenden Zyklus $(s_z, s_z, \ldots)$ wird die Gesamtbelohnung in diesem Zustand wie folgt gebildet:

$$\begin{aligned} Q^{Zyklus}(s_z, a_z) &:= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot \left(\widetilde{h}(s_z) + \gamma \cdot \widetilde{h}(s_z) + \ldots \right) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot \widetilde{h}(s_z) \cdot (1 + \gamma + \ldots) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot (\widetilde{h}(s_z)) \cdot \sum_{t=0}^{\infty} (\gamma)^t \end{aligned} \tag{4.16}$$

Für einen Zyklus, der aus zwei Schritten $(s_z, s_{z+1}, s_z, \ldots)$ besteht, gilt:

$$\begin{aligned} Q^{Zyklus}(s_z, a_z) &:= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot \left(\widetilde{h}(s_z) + \gamma \cdot \widetilde{h}(s_{z+1}) + \gamma^2 \cdot \widetilde{h}(s_z) + \ldots\right) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot (\widetilde{h}(s_z) + \gamma \cdot \widetilde{h}(s_{z+1})) \cdot \left(1 + \gamma^2 + \gamma^4 + \ldots\right) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot (\widetilde{h}(s_z) + \gamma \cdot \widetilde{h}(s_{z+1})) \cdot \sum_{t=0}^{\infty} (\gamma^2)^t \end{aligned} \tag{4.17}$$

Für einen Zyklus, der aus P Schritten $(s_z, s_{z+1}, \ldots s_{z+P-1}, s_z, \ldots)$ besteht, gilt:

$$\begin{aligned} Q^{Zyklus}(s_z, a_z) &:= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot \left(\widetilde{h}(s_z) + \gamma \cdot \widetilde{h}(s_{z+1}) + \ldots + \gamma^{P+P-1} \cdot \widetilde{h}(s_{z+P-1})\right) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot (\widetilde{h}(s_z) + \ldots \gamma^{P-1} \cdot \widetilde{h}(s_{z+P-1})) \cdot \left(1 + \gamma^P + \ldots\right) \\ &= -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot (\widetilde{h}(s_z) + \ldots \gamma^{P-1} \cdot \widetilde{h}(s_{z+P-1})) \cdot \sum_{t=0}^{\infty} (\gamma^P)^t \end{aligned} \tag{4.18}$$

Es wird angenommen, dass eine Strategie π^*, die den Agenten zum Ziel führt, für das untersuchte Problem vorhanden ist und T Schritte beinhaltet. Diese Kette wird als $S^{\pi^*} := \{(s_0, a_0), \ldots (s_T, a_T)\}$ bezeichnet. Die Werte der heuristischen Funktion an den besuchten Zuständen $h(s_0), \ldots, h(s_T)$ seien bekannt. Die verwendete heuristische Funktion ist als parametrisierte normierte Abstandsfunktion zum Zielzustand definiert, $h(s_i) = k \cdot \widetilde{h}(s_i)$, wobei k der Einflussparameter ist und der normierte Abstand $\widetilde{h}(s_i) \forall s_i \in S$ zum Zielzustand ist.
Durch Einsetzen der Umformung (4.18) in die Ungleichung (4.15) wird Ungleichung (4.19) erzielt. Die vorhandene Strategie, die den Agenten zum Zielzustand führt, wird vom Zustand der Zykluskandidaten definiert, d. h. $s_{Start} := s_z$. Jeder während der Mission angefahrene Zustand s_{i+1} wird durch Ausführen der Aktion $a_i = \pi^*(s_i)$ im Zustand s_i für $i = 0, \ldots, T-1$, die von der vorhandenen Strategie π^* vorgegeben wird, bestimmt.

$$\begin{aligned} Q^{\pi^*}(s_z, \pi^*(s_z)) &:= -1 \cdot \sum_{t=0}^{T} \gamma^t + k \cdot \sum_{t=0}^{T} \gamma^t \widetilde{h}(s_t) \overset{z.Z}{>} \\ &\overset{z.Z}{>} -1 \cdot \sum_{t=0}^{\infty} \gamma^t + k \cdot \widetilde{\Delta}(\gamma) \cdot \sum_{t=0}^{\infty} (\gamma^P)^t \overset{(4.18)}{=:} Q^{Zyklus}(s_z, \pi^{Zyklus}(s_z)) \end{aligned} \tag{4.19}$$

wobei $\widetilde{\Delta}(\gamma) := \widetilde{h}(s_z) + \ldots + \gamma^{P-1} \cdot \widetilde{h}(s_{z+P-1})$ und P die Anzahl der Schritte im Zyklus ist. Für die theoretischen Überlegungen wird das Standard-Belohnungsmodell angewendet: Für jeden Schritt ist der Bestrafungswert auf -1 gesetzt, außer wenn der Zielzustand erreicht wird. Wenn der Zielzustand erreicht wird, wird der Agent eine Belohnung von Null (also keine Bestrafung) erhalten. Die durchgeführte Beweiskette kann mit wenig Aufwand auf den Fall mit dem Belohnungsmodell mit Sonderbelohnung angewendet werden, vergleiche auch Bemerkung 4.3.3.
Wenn die Ungleichung (4.19) nach k in Abhängigkeit von γ und T aufgelöst wird, kann das gesuchte zulässige Intervall für den heuristischen Einflussparameter k aufgestellt werden. Dieser Schritt wird im Beweis zu Satz 4.4.1 durchgeführt. Der Beweis wird für das Gitterweltproblem und für das als normierte und parametrisierte Abstandsfunktion zum Ziel definierte heuristische Funktion ausgelegt. Die vorge-

nommenen Abschätzungen können aber mit einem entsprechenden Diskretisierungsschritt auch für das Navigationsproblem und das MCP angewendet werden, da auch hier die angewendete heuristische Funktion in allen drei Problemarten die gleiche Heuristik realisiert.

Satz 4.4.1. *Für die Garantie für das Nicht-Entstehen von Zyklen an den Zykluskandidaten $s_<, s_>$ während der Anwendung des SARSA*-Algorithmus soll der Parameter k in Abhängigkeit vom Diskontierungsfaktor $0 < \gamma \leq 1$ und von den für den Zykluskandidaten im Vorfeld ermittelten Parametern $T_>, T_<$ bestimmt werden, so dass Ungleichung (4.20) gilt. Diese Ungleichung repräsentiert das gesuchte zulässige Intervall.*

$$\frac{\gamma^{T_<+1}}{(\gamma^{T_<+1} + \frac{\widetilde{\Delta}_<(\gamma)\cdot(1-\gamma)}{1-\gamma^P}) - 1} \leq k \leq \gamma^{T_>+1} \tag{4.20}$$

Die Zykluskandidaten können wie folgt bestimmt werden. Für $k > 0$ entsteht ein Zykluskandidat $s_>$ in einem am weitesten vom Zielzustand entfernten Zustand. Von diesem Zustand aus werden $T_>$ Schritte bis zum Erreichen des Ziels benötigt, wenn die bekannte Strategie verfolgt wird. Für $k < 0$ kann der Zykluskandidat $s_<$ dann entstehen, wenn $\widetilde{\Delta}_<(\gamma) \leq (1 - \gamma^{T_<+1}) \cdot \frac{(1-\gamma^P)}{1-\gamma}$ gilt, wobei der am Zykluskandidaten $s_<$ potenziell entstehende Zyklus P Schritte beinhaltet und $\widetilde{\Delta}_<(\gamma)$ zur $\widetilde{\Delta}_<(\gamma) := \widetilde{h}(s_<) + \gamma \cdot \widetilde{h}(s_{<+1}) + \ldots \gamma^{P-1} \cdot \widetilde{h}(s_{<+P})$ berechnet wird. Für den Zykluskandidaten $s_<$ werden $T_<$ Schritte notwendig bis der Konflikt der verfolgten Strategie mit der heuristischen Funktion gelöst ist.

Bemerkung 4.4.4. *Dieser Satz ist eine hinreichende Bedingung für die Garantie für das Nicht-Entstehen von Zyklen in den Zykluskandidaten bei der Einbindung der heuristischen Information in den Lernprozess (SARSA*-Algorithmus). Es wird angenommen, dass der Standard-SARSA-Algorithmus zu einer Lösung konvergiert. Aus der Sicherstellung des Nicht-Entstehens der Zyklen bei der Anwendung des SARSA*-Algorithmus wird intuitiv die Konvergenz zur gesuchten Strategie, die den Agenten zum Ziel steuert, angenommen. Aus der Konvergenz kann aber die Beschleunigung der Lerngeschwindigkeit und der Lernqualität nicht abgeleitet werden.*

Bemerkung 4.4.5. *Ein Beispiel für die genannten Bezeichnungen für das Szenario mit Weggabelung wäre $s_< := s_1$ mit $T_< := 4$, $s_> = s_{11}$ mit $T_> = 5$, $Norm = \frac{1}{\sqrt{13}}$.*
Es ist schwierig die genauen Werte für $T_<$ bzw. $T_>$ und den Wert $Norm$ für beide Verfahren zu bestimmen, da das Umgebungsmodell für das Problem nicht vorhanden ist. Diese Werte können aber abgeschätzt werden. Die ungefähr befahrbare Fläche ist normalerweise schon im Vorfeld bekannt. Es könnten unvorteilhafte Szenarien aufgestellt werden, die nicht der Wahrheit entsprechen müssen, aber für die Bestimmung des zulässigen Intervalls als vorsichtige Prognose relevant sind.

Bemerkung 4.4.6. *Wenn kein Zykluskandidat im Fall $k < 0$ existiert, strebt der untere Grenzwert ins Unendliche. Für $k > 0$ existiert der Zykluskandidat mit der oben definierten heuristischen Funktion am Zustand $s \in S$ mit $\widetilde{h}(s) = 1.0$ immer, da die so definierte heuristische Funktion in Konflikt zur Problemstellung steht.*[27]

Beweis. Durch Auflösung und Approximation der aufgestellten Ungleichung (4.19) wird der Parameter k in Abhängigkeit von $(\widetilde{\Delta}_<, T_<, T_>, \gamma)$ bestimmt.
Es gilt $\forall s \in S$ die Abschätzung

$$0 \leq \widetilde{h}(s) \leq 1. \tag{4.21}$$

[27] Der beste Zustand liegt nach angenommener Heuristik am weitesten entfernt vom Zielzustand.

Hier werden zwei Fälle in Betracht gezogen: $k > 0$ und $k < 0$.
Sei zunächst $k < 0$. $T_<$ ist die Anzahl der Schritte vom Zykluskandidaten $s_<$, wo die heuristische Funktion in Konflikt zur verfolgten Strategie steht, bis zum Erreichen des Ziels unter der Anwendung der verfolgten Strategie π^*. Sei P die Anzahl der Schritte vom möglichen Zyklus, $\Delta_<(\gamma) = h(s_<) + \ldots + \gamma^{P-1} h(s_{<+P-1})$ die Abschätzung des heuristischen Einflusses im Zyklus. Die Ungleichung (4.19) kann wie folgt durch Multiplikation mit -1 umformuliert werden:

$$\sum_{t=0}^{T_<} \gamma^t + \underbrace{-1 \cdot k}_{\geq 0 \text{ da } k<0} \cdot \sum_{t=0}^{T_<} \gamma^t \widetilde{h}(s_t) \overset{z.Z.}{\leq} \sum_{t=0}^{\infty} \gamma^t + -1 \cdot k \cdot \widetilde{\Delta}_<(\gamma) \sum_{t=0}^{\infty} (\gamma^P)^t$$

wobei der optimale Weg angefangen vom vermuteten Zykluskandidaten gebildet wird, also $s_{t=0} := s_z$. Mit der Abschätzung (4.21) für den heuristischen Anteil gilt:

$$\begin{aligned} \underbrace{\sum_{t=0}^{T_<} \gamma^t + |k| \cdot \sum_{t=0}^{T_<} \gamma^t \widetilde{h}(s_t)}_{A} &\overset{(4.21)}{\leq} \underbrace{\sum_{t=0}^{T_<} \gamma^t + |k| \cdot \sum_{t=0}^{T_<} \gamma^t}_{B} \\ &\overset{z.Z.}{\leq} \underbrace{\sum_{t=0}^{\infty} \gamma^t + |k| \cdot \widetilde{\Delta}_<(\gamma) \cdot \sum_{t=0}^{\infty} (\gamma^P)^t}_{C} \end{aligned} \tag{4.22}$$

Wenn der Parameter k in Abhängigkeit von γ so bestimmt wird, dass der Fall $B \leq C$ gilt, dann gilt mit einem so definierten Parameter k auch der Fall $A \leq C$.
Sei $0 < \gamma < 1$.

$$\begin{aligned} &\sum_{t=0}^{T_<} \gamma^t + |k| \cdot \sum_{t=0}^{T_<} \gamma^t \overset{z.Z.}{\leq} \sum_{t=0}^{\infty} \gamma^t + |k| \cdot \widetilde{\Delta}_<(\gamma) \cdot \sum_{t=0}^{\infty} (\gamma^P)^t \qquad \div \sum_{t=0}^{\infty} \gamma^t \\ &\iff \\ &\underbrace{\frac{\sum_{t=0}^{T_<} \gamma^t}{\sum_{t=0}^{\infty} \gamma^t}}_{\overset{Bem.\ 4.4.3}{:=} 1-\gamma^{T_<+1}} + |k| \cdot \underbrace{\frac{\sum_{t=0}^{T_<} \gamma^t}{\sum_{t=0}^{\infty} \gamma^t}}_{:=1-\gamma^{T_<+1}} \overset{z.Z.}{\leq} 1 + |k| \cdot \widetilde{\Delta}_<(\gamma) \cdot \underbrace{\frac{\sum_{t=0}^{\infty} (\gamma^P)^t}{\sum_{t=0}^{\infty} \gamma^t}}_{\overset{Bem.\ 4.4.1}{:=} \frac{1-\gamma}{1-\gamma^P}} \\ &\iff \\ &|k| \cdot \underbrace{\left(1 - \gamma^{T_<+1} - \widetilde{\Delta}_<(\gamma) \cdot \frac{1-\gamma}{(1-\gamma^P)}\right)}_{=1-\gamma^{T_<+1} - \frac{\widetilde{\Delta}_<(\gamma)(1-\gamma)}{1-\gamma^P}} \overset{z.Z.}{\leq} \underbrace{1 - 1 + \gamma^{T_<+1}}_{=\gamma^{T_<+1}} \end{aligned} \tag{4.23}$$

Aus der Ungleichung (4.23) entsteht die Ungleichung (4.24). Wenn diese Ungleichung gilt, kann garantiert werden, dass keine Zyklen entstehen.

$$|k| \cdot (1 - \gamma^{T_<+1} - \frac{\widetilde{\Delta}_<(\gamma)(1-\gamma)}{1-\gamma^P}) \leq \gamma^{T_<+1} \tag{4.24}$$

Da $0 < \gamma \leq 1$, ist die rechte Seite der Ungleichung (4.24) immer positiv ($\gamma^{T_<+1} \geq 0$). Wenn für jeden Zustand $s_t \in S$ folgendes gilt $\widetilde{\Delta}_<(\gamma) := h(s_t) + \gamma \cdot h(s_{t+1} + \ldots + \gamma^{P-1} \cdot h(s_{t+P-1}) > (1 - \gamma^{T_<+1}) \cdot \frac{(1-\gamma^P)}{1-\gamma}$, dann ist die linke Seite der Ungleichung immer kleiner als Null, da $1 - \gamma^{T_<+1} <$

$\frac{\widetilde{\Delta}_<(\gamma)\cdot(1-\gamma)}{1-\gamma^P}$. Also kann unabhängig von der Wahl des Parameters $k < 0$ garantiert werden, dass keine Zykluskandidaten entstehen. Wenn jedoch solche Zustand-Aktions-Paare existieren, dass $\widetilde{\Delta}_<(\gamma) \leq (1-\gamma^{T_<+1})\frac{(1-\gamma^P)}{1-\gamma}$ gilt, dann muss der Parameter k in Abhängigkeit vom Parameter γ so gewählt werden, dass die Bedingung (4.24) gilt, also $|k| \leq \frac{\gamma^{T_<+1}}{1-\gamma^{T_<+1}-\frac{\widetilde{\Delta}_<(\gamma)\cdot(1-\gamma)}{1-\gamma^P}}$.

Somit soll gelten:

$$\frac{\gamma^{T_<+1}}{\gamma^{T_<+1}+\frac{\widetilde{\Delta}_<(\gamma)(1-\gamma)}{1-\gamma^P}-1} \leq k < 0, \text{ für } \widetilde{\Delta}_<(\gamma) \leq (1-\gamma^{T_<+1})\frac{(1-\gamma^P)}{(1-\gamma)} \tag{4.25}$$

Für den Fall $\gamma = 1$:

$$\begin{aligned} \underbrace{\lim_{\gamma\to 1}\frac{\sum_{t=0}^{T_<}\gamma^t}{\sum_{t=0}^{\infty}\gamma^t}}_{\overset{\gamma\to 1}{\to} 0} + |k| \cdot \underbrace{\lim_{\gamma\to 1}\frac{\sum_{t=0}^{T_<}\gamma^t}{\sum_{t=0}^{\infty}\gamma^t}}_{\overset{\gamma\to 1}{\to} 0} &\overset{z.Z.}{\leq} 1 + |k| \cdot \widetilde{\Delta}_<(\gamma) \cdot \underbrace{\lim_{\gamma\to 1}\frac{\sum_{t=0}^{\infty}(\gamma^P)^t}{\sum_{t=0}^{\infty}\gamma^t}}_{\overset{Bem.\ 4.4.1}{:=} \frac{1-\gamma}{(1-\gamma^P)} \overset{\gamma\to 1}{\to} \frac{1}{2}} \\ &\iff \\ 0 &\leq 1 + |k| \cdot \widetilde{\Delta}_<(\gamma) \cdot \frac{1}{2} \end{aligned} \tag{4.26}$$

Somit gilt die Ungleichung (4.26) immer, da die linke Seite der Ungleichung Null ist, und auf der rechten Seite eine positive Zahl steht. Also ist der Parameter k im Fall $\gamma = 1$ nicht nach unten begrenzt.
Sei jetzt $k > 0$. Für diesen Fall steht die Heuristik komplett im Widerspruch zur Aussage der verfolgten Strategie, so dass der komplette Weg vom Zykluskandidaten bis zum Ziel zum Ausstieg aus dem Widerspruch benötigt wird. $T_>$ ist somit die Anzahl der Schritte bis zum Erreichen des Ziels, ausgehend vom Zykluskandidaten $s_>$. Sei P die Anzahl der Schritte im möglichen Zyklus, angefangen vom Zykluskandidaten $s_>$ und $\widetilde{\Delta}_> := \widetilde{h}(s_>) + \gamma \cdot \widetilde{h}(s_>+1) + \ldots \gamma^{P-1} \cdot \widetilde{h}(s_>+P-1)$. Die Ungleichung (4.19) soll gelten, um garantieren zu können, dass der Agent nicht das Verbleiben im Zyklus bevorzugt.

$$\begin{aligned} \underbrace{-1 \cdot \sum_{t=0}^{T_>}\gamma^t + k \cdot \sum_{t=0}^{T_>}\gamma^t\widetilde{h}(s_t)}_{A} &\overset{(4.21)}{\geq} \underbrace{-1 \cdot \sum_{t=0}^{T_>}\gamma^t + k \cdot \sum_{t=0}^{T_>}\gamma^t \cdot 0}_{B} \\ &\overset{z.Z.}{\geq} \underbrace{-1 \cdot \sum_{t=0}^{\infty}\gamma^t + k \cdot \widetilde{\Delta}_>(\gamma) \cdot \sum_{t=0}^{\infty}(\gamma^P)^t}_{C} \end{aligned} \tag{4.27}$$

Wenn die Parameter für (k, γ) so bestimmt werden, dass die Ungleichung $B \geq C$ gilt, dann wird auch die aufgestellte Ungleichung $A \geq C$ gelten. Wenn die heuristische Funktion wie im Vorschlag (4.4) definiert wird, dann existieren für den Fall $k > 0$ solche Zykluskandidaten immer. An der am weitesten entfernten Stelle gilt die Ungleichung (4.28).

$$\widetilde{\Delta}_>(\gamma) \overset{(4.4)}{\leq} \underbrace{1}_{\widetilde{h}(s_>)} + \gamma \cdot 1 + \ldots + \gamma^{P-1} \cdot 1 \tag{4.28}$$

Wenn die Abschätzung (4.28) für den Wert $\widetilde{\Delta}(\gamma)_>$ gilt, wird folgende Umformung gelten:

$$\frac{\gamma^{T_>+1}\cdot(1-\gamma^P)}{\widetilde{\Delta}_>(\gamma)\cdot(1-\gamma)} \overset{(4.28)}{\geq} \frac{\gamma^{T_>+1}\cdot \underbrace{(1-\gamma^P)}_{=(1-\gamma)\cdot(\gamma^{P-1}+\gamma^{P-2}+\ldots+1)}}{(1+\gamma+\ldots\gamma^{P-1})\cdot(1-\gamma)} = \gamma^{T_>+1} \tag{4.29}$$

Also ist der Parameter k immer nach oben begrenzt.

$$\begin{aligned} -1\cdot\sum_{t=0}^{T_>}\gamma^t &\overset{z.Z.}{\geq} -1\cdot\sum_{t=0}^{\infty}\gamma^t + k\cdot\widetilde{\Delta}_>(\gamma)\cdot\sum_{t=0}^{\infty}(\gamma^P)^t \\ &\Longleftrightarrow \\ k\cdot\widetilde{\Delta}_>(\gamma)\cdot\underbrace{\sum_{t=0}^{\infty}(\gamma^P)^t}_{\overset{Bem.\ 4.4.1}{=}\frac{1}{1-\gamma^P}} &\overset{z.Z.}{<} \underbrace{\frac{\gamma^{T_>+1}}{1-\gamma}}_{Bem.\ 4.4.2} \\ &\Longleftrightarrow \\ 0<k\leq\gamma^{T_>+1} &\overset{(4.29)}{\leq} \frac{\gamma^{T_>+1}\cdot(1-\gamma)^P}{\widetilde{\Delta}_>(\gamma)\cdot(1-\gamma)} \end{aligned} \tag{4.30}$$

□

4.4.4 Bestimmung der Zykluskandidaten

An dieser Stelle wird genauer auf die Bedingungen eingegangen, unter denen Zykluskandidaten im SARSA*-Algorithmus entstehen. Es wird erläutert, wie die Zykluskandidaten erkannt werden können, und welche Bedingungen die Entstehung von Zyklen verhindern.
In Satz 4.4.1 gilt die Abschätzung (4.20), die vom Aufbau der heuristischen Funktion und dem Umgebungsmodell abhängig ist. Dabei können zwei Arten der Zykluskandidaten entstehen, für $k<0$ der Zykluskandidat $s_<$ und für $k>0$ der Zykluskandidat $s_>$. Zur Anwendung des Satzes ist zumindest eine grobe Struktur des Umgebungsmodells notwendig, um die Werte $\widetilde{\Delta}_<(\gamma)$, $T_<$ und $T_>$ abschätzen zu können. In diesem Kapitel wird auf die Bedeutung der im Satz aufgestellten Bedingungen näher eingegangen.
Wenn der heuristische Einflussparameter einen positiven Wert annimmt ($k>0$), dann existiert immer ein Zykluskandidat, wenn gleichzeitig die heuristische Funktion als normierte Distanz zum Ziel definiert ist. Die heuristische Funktion steht in diesem Fall in Konflikt zur eigentlichen Aufgabenstellung, da die verfolgte Heuristik "je weiter sich der Agent vom Zielzustand befindet, desto besser ist seine Lage" lautet. Der gesuchte Zykluskandidat $s_>$ ist der Zustand, der am weitesten vom Ziel entfernt liegt.
Für den Fall $k<0$ ist die Fragestellung der Existenz von Zykluskandidaten komplexer. Für manche Umgebungen steht die heuristische Funktion nie in Konflikt zur verfolgten Strategie. Für andere Umgebungen treten diese Konflikte auf. In diesem Fall entstehen Zykluskandidaten, die sich zu Zyklen entwickeln können. Der aufgestellte Satz 4.4.1 beantwortet die Frage, an welchen Stellen Zykluskandidaten entstehen können und unter welchen Bedingungen keine Zyklen an den Stellen der Zykluskandidaten entstehen.

$$\widetilde{\Delta}_<(\gamma)\leq(1-\gamma^{T_<+1})\cdot\frac{(1-\gamma^P)}{1-\gamma} \tag{4.31}$$

Ein Zustand $s_<$ ist ein Zykluskandidat, wenn für vorgegebene γ, $\widetilde{\Delta}_<(\gamma)$, $T_<$ und P die Ungleichung (4.31) gilt, wobei $T_<$ die Anzahl der notwendigen Schritte bis zur Lösung des Konflikts ist.
Anhand eines Beispiels wird die Abhängigkeit der Entstehung der möglichen Zykluskandidaten vom Parameter γ erläutert. Das Beispiel wird für das Gitterweltproblem im Szenario Umgebung mit Weggabelung, siehe Abbildung 4.2(a), durchgeführt und dient der Deutung der Ungleichung (4.31).
Für $k < 0$ können für bestimmte Werte der Parameter γ und k zwei mögliche Zyklen im Zustand s_1 entstehen. Die unterschiedlichen Zyklen entstehen für unterschiedliche Belohnungsmodelle. Der Zyklus mit zwei Schritten $(s_1, s_5, s_1, s_5, \ldots)$ entsteht bei Anwendung des Belohnungsmodells mit Sonderbestrafung für Kollisionen. Der Zyklus mit einem Schritt $(s_1, s_1, \ldots)$ entsteht bei Anwendung des Standard-Belohnungsmodells. An dieser Stelle ist in dieser Umgebung der Wert $\widetilde{\Delta}_<(\gamma)$ in Abhängigkeit vom Wert γ wie folgt definiert: $\Delta_<(\gamma) := k \cdot \widetilde{\Delta}_<(\gamma) = k \cdot (\frac{1}{\sqrt{13}} + \gamma \cdot \sqrt{\frac{2}{13}})$ bzw. $\Delta_< := k \cdot \widetilde{\Delta}_< = k \cdot \frac{1}{\sqrt{13}}$.
Der Zykluskandidat kann erst dann entstehen, wenn die Ungleichung $\widetilde{\Delta}_<(\gamma) < (1 - \gamma^{T_<+1})\frac{1-\gamma^2}{1-\gamma}$ bzw. $\widetilde{\Delta}_< < (1 - \gamma^{T_<+1})$ gilt, da an dieser Stelle ein Konflikt zwischen den Aussagen der verfolgten Strategie und der heuristischen Funktion entsteht. Im ersten Fall hängt $\widetilde{\Delta}_<(\gamma)$ vom Parameter γ ab. Wenn die Gerade unterhalb der mit $T_<$ parametrisierten Kurven $K_{T_<}(\gamma) := (1 - \gamma^{T_<+1}) \cdot (1 + \gamma)$ liegt, gilt die Ungleichung, und Zykluskandidaten können entstehen, vergleiche blaue Gerade in Abbildung 4.6. Im zweiten Fall ist $\widetilde{\Delta}_<$ ein konstanter Wert.
$T_<$ kann unterschiedlich definiert sein. Für $T_<$ kann eine optimistische Einschätzung getroffen werden.

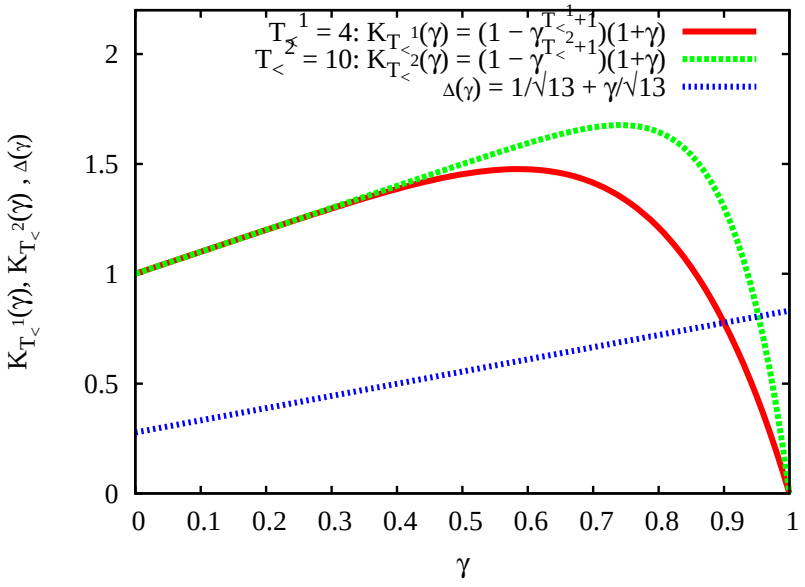


Abbildung 4.6: Auswirkung des Diskontierungsfaktors γ auf die Entstehung der Zykluskandidaten im Fall $k \leq 0$. Wenn der Parameter γ groß genug gewählt ist, und die Anzahl der für den Ausstieg aus dem Zykluskandidaten benötigten Schritte klein ist, besteht keine Gefahr für die Entstehung von Zyklen. Für die γ-Werte, bei denen die Gerade $\delta(\gamma)$ oberhalb der Kurven $K_{T^1_<}(\gamma)$ bzw. $K_{T^2_<}(\gamma)$ liegt, entstehen keine Zyklen für $k \leq 0$.

In diesem Beispiel wird $T^1_<$ auf den Wert 4 gesetzt, da diese Anzahl der Schritte vom Zustand s_1 bis zum Zustand s_{11} benötigt wird. Ab dem Zustand s_{11} steht die Heuristik nicht in Konflikt zur verfolgten Strategie. Der Wegabschnitt vom Zustand s_1 bis zum Zustand s_{11} benötigt die folgenden vier Schritte: $\{(s_1, \downarrow), (s_5, \rightarrow), (s_6, \rightarrow), (s_7, \downarrow)\}$. Es kann eine vorsichtigere Einschätzung für den Wert $T_<$ getroffen werden. In diesem Fall wird der Wert $T_<$ als die Anzahl der Schritte von s_1 bis zum Erreichen des Ziels

s_0 bei Anwendung der optimalen Strategie gewählt werden, $T^2_< = 10$.
Aus Abbildung 4.6 wird ersichtlich, dass für $\gamma > 0.91$ im Fall $T^1_< = 4$ bzw. $\gamma > 0.96$ im Fall $T^2_< = 10$ keine Zyklen entstehen können. Für $T^2_< = 10$ und $\gamma > 0.96$ liegt die Gerade $\widetilde{\Delta}_<(\gamma)$ oberhalb der Kurve $K_{T^2_<}(\gamma)$. Für $T^1_< = 4$ und $\gamma > 0.91$ liegt die Gerade $\widetilde{\Delta}_<(\gamma)$ oberhalb der Kurve $K_{T^1_<}(\gamma)$. Für auf diese Weise eingesetzte Parameter $\gamma > 0.91$ bzw. $\gamma > 0.96$ und $k < 0$ gilt stets die Ungleichung (4.19).
Im Fall $k < 0$ entstehen keine Zyklen an den Zykluskandidaten bei Anwendung des SARSA*-Algorithmus, wenn der Diskontierungsfaktor entsprechend groß gewählt wird, und die Anzahl der bis zum Ausstieg aus dem Zyklus benötigten Schritte $T_<$ klein ist. Wenn die Ungleichung (4.31) für alle Zykluskandidaten und das bestimmte γ nie gilt, darf der Parameter $k(\gamma)$ beliebige Werte im Intervall $[-\infty, 0]$ annehmen.

4.4.5 Zulässiges Intervall für das Gitterweltproblem

Die theoretisch und manuell bestimmten Grenzwerte für das zulässige Intervall des heuristischen Einflussparameters k werden in Abhängigkeit vom Diskontierungsfaktor γ für zwei Szenarien im Gitterweltproblem miteinander verglichen.
Die manuelle Bestimmung der Grenzwerte für das zulässige Intervall des Parameters k ist in Kapitel 4.3.2 genauer erläutert. Wenn für einen vorgegebenen Parameter γ der bestimmte Parameter k den manuell bestimmten Grenzwert unterschreitet (für $k < 0$) bzw. überschreitet (für $k > 0$), divergiert der SARSA*-Algorithmus, es entstehen also Zyklen an den Stellen der Zykluskandidaten.
Für die Bestimmung der theoretischen Grenzwerte wird die folgende Parametrisierung für das Szenario mit Weggabelung gewählt. Für $k < 0$ liegt der Zykluskandidat im Zustand s_1. Die Anzahl der Schritte von s_1 bis s_{11} ist $T_< = 4$, da ab dem Zustand s_{11} die Aussage der Heuristik der Aussage der optimalen Strategie nicht widerspricht.
Für den Fall $k > 0$ ist die Anzahl der Schritte für den Weg vom Zykluskandidaten s_{11} bis zum Zielzustand s_0 bei Anwendung der optimalen Strategie $T_> = 5$, wobei der Zykluskandidat $s_> := s_{11}$ der vom Zielzustand am weitesten entfernt liegende Zustand ist.
Die Grenzwerte für die Umgebung mit Weggabelung werden an dieser Stelle miteinander verglichen. In Abbildung 4.7 sind die manuell und theoretisch bestimmten Grenzwerte für den Parameter k in Abhängigkeit vom Diskontierungsfaktor γ dargestellt. Wenn der Diskontierungsfaktor einen niedrigen Wert annimmt, soll der Parameter k so nah wie möglich am Wert Null liegen, sonst kann der Widerspruch zwischen der heuristischen Information und dem tatsächlichen Umgebungsmodell in Bezug auf die Information über die Nähe zum Ziel zur Divergenz des Algorithmus führen, siehe Abbildung 4.7.
Die Wahl des Diskontierungsfaktors wirkt sich signifikant auf die Breite des zulässigen Intervalls aus. Je kleiner der Parameter γ gewählt wird, desto schmaler ist das zulässige Intervall für den Parameter k. Z. B. ist für $\gamma = 1.0$ das zulässige Intervall $k \in [-\infty; 1.0]$ und für $\gamma = 0.6$ liegt dieses manuell bestimmte Intervall ungefähr bei $k \in [-0.089; 0.498]$, das theoretisch bestimmte Intervall ist $k \in [\underbrace{-0.084}_{:=\frac{0.6^5}{0.6^5-1.0}}, \underbrace{0.05}_{=0.6^7}]$.
Dies lässt sich dadurch erklären, dass mit einem kleiner werdenden γ der Agent immer kurzsichtiger wird. Die zukünftige Belohnung hat dadurch für den Agenten eine immer kleinere Bedeutung. Der Agent wird gierig und schaut nur auf das, was einen sofortigen Nutzen bringt. Wenn nur die momentane Belohnung berücksichtigt wird, bleibt der Agent in den Zyklen stecken und wird das tatsächliche Ziel nicht mehr erreichen.
In Abbildung 4.7 kann eine gute Approximation der manuell bestimmten Grenzwerte durch die theoretisch bestimmten Grenzwerte beobachtet werden.

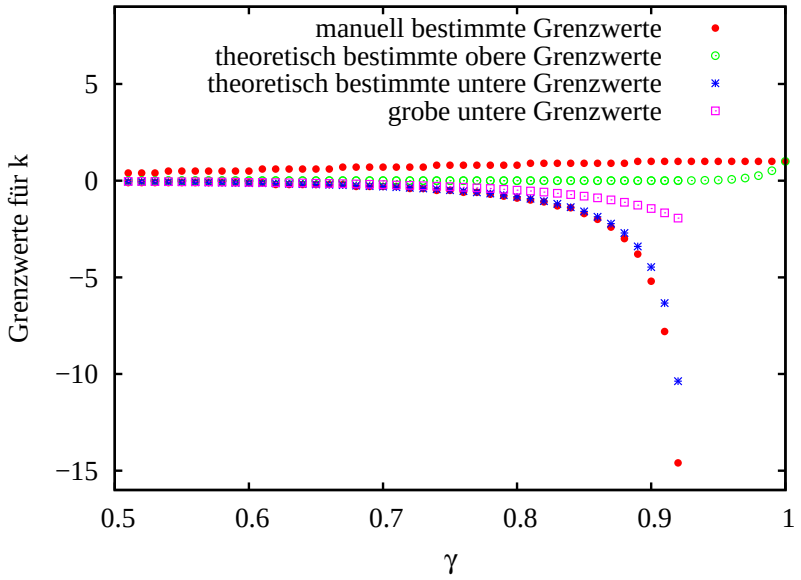


Abbildung 4.7: Abhängigkeit der Grenzwerte des zulässigen Intervalls für den Parameter k vom Diskontierungsfaktor γ für die Umgebung mit Weggabelung in der Gitterwelt, manuell bzw. theoretisch berechnet. $T_> = 5$ ist die Anzahl der Schritte vom Zustand s_{11} bis zum Zielzustand s_0. $T_< = 4$ ist die Anzahl der Schritte vom Zustand s_1 bis zum Zustand s_{11}.

Für das andere Szenario im Gitterweltproblem, die Umgebung mit Sackgassen, werden die gleichen

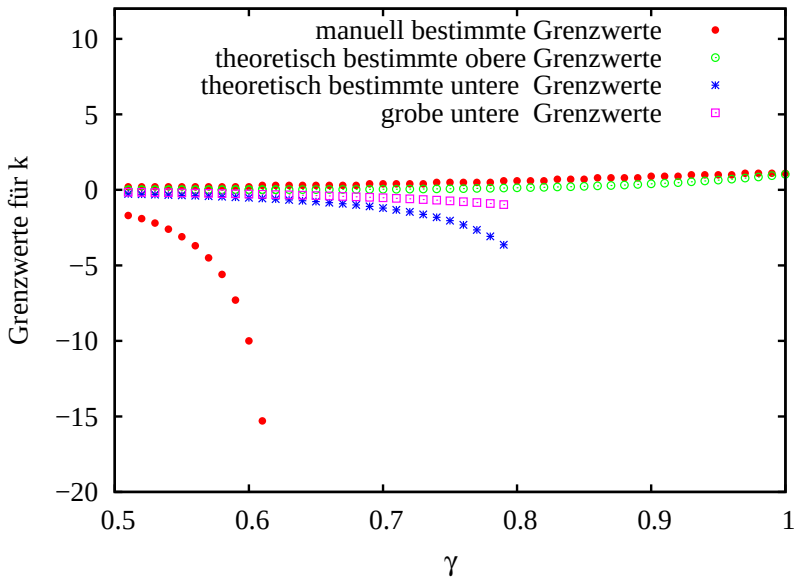


Abbildung 4.8: Abhängigkeit der Grenzwerte des zulässigen Intervalls für den Parameter k vom Diskontierungsfaktor γ für die Umgebung mit Sackgassen in der Gitterwelt, manuell bzw. theoretisch berechnet. $T_< = 2$ ist die Anzahl der Schritte vom Zustand s_2 bis zum Zustand s_{14}. $T_> = 9$ ist die Anzahl der Schritte vom Zustand s_{17} bis zum Zustand s_0.

Vorbereitungen wie für die Umgebung mit Weggabelung getroffen. Für einen negativen Wert des heuristischen Einflussparameters $k < 0$ wird der Zustand s_2 zu einem Zykluskandidaten, dabei ist die Anzahl

der Schritte von Zustand s_2 bis zum Zustand s_{14} gleich, $T_< = 2$. Dieser Teilweg beinhaltet folgende zwei Schritte $(s_2, \downarrow), (s_8, \downarrow)$. Ab Zustand s_{14} widerspricht die heuristische Funktion den Aussagen der optimalen Strategie nicht mehr. Für den Fall $k > 0$ ist der Zykluskandidat der Zustand s_{17}, der am weitesten vom Zielzustand s_0 liegt. Der Parameter $T_>$, der als die Anzahl der Schritte vom Zustand s_{17} bis zum Zustand s_0 bei Anwendung der optimalen Strategie definiert ist, wird zu $T_> = 9$ gesetzt.
In Abbildung 4.8 sind die Grenzwerte für das zulässige Intervall für Parameter k für unterschiedliche Werte des Diskontierungsfaktors γ dargestellt. Die manuell und theoretisch bestimmten, oberen Grenzwerte liegen sehr nah beieinander. Für den unteren Grenzwert ($k < 0$) liegen die manuell und theoretisch bestimmten Grenzwerte weit auseinander. Der theoretische Grenzwert erfüllt aber seinen Zweck und liegt immer oberhalb des manuell bestimmten Grenzwertes für das zulässige Intervall des heuristischen Einflussparameters k. Die theoretisch bestimmten Grenzwerte sind nur eine Abschätzung der benötigten zulässigen Intervalle für den heuristischen Einflussparameter k.
Für zwei Umgebungen im Gitterweltproblem wurde gezeigt, dass das theoretisch bestimmte, zulässige Intervall für den Parameter k innerhalb des manuell bestimmten zulässigen Intervalls liegt. Die theoretisch und manuell bestimmten Grenzwerte des Intervalls können zwar weit auseinander liegen, die theoretisch bestimmten Grenzwerte erfüllen aber die gestellte Aufgabe und liefern ein korrektes zulässiges Intervall für den heuristischen Einflussparameter k in Abhängigkeit vom Diskontierungsfaktor γ.

4.4.6 Grobe Abschätzung des zulässigen Intervalls

Die Abschätzung (4.20) hängt von den Werten $\Delta_<(\gamma)$, $T_<$ und $T_>$ ab. Diese Abschätzung kann ohne die Unbekannte $\Delta_<(\gamma)$ und somit ohne das Wissen des Umgebungsmodells etwas gröber bestimmt werden. Dabei wird angenommen, dass die Zykluskandidaten im extremen Punkt auftreten.
Für den Fall $k < 0$ wird die $\widetilde{\Delta}_<(\gamma)$ durch $0 := 0 + 0 \cdot \gamma$ approximiert. Da in diesem Fall immer $\widetilde{\Delta}_<(\gamma) := 0 \leq \underbrace{(1 - \gamma^{T_<+1})}_{\geq 0} \cdot (1 + \gamma)$ gilt, existiert immer ein Zykluskandidat. Für den Fall $k > 0$ ist in der Abschätzung aus dem Satz keine Abhängigkeit von $\Delta_>(\gamma)$ vorhanden.

$$\frac{\gamma^{T_<+1}}{\gamma^{T_<+1} - 1} \leq k \leq \gamma^{T_>+1} \tag{4.32}$$

In Abbildung 4.9 ist eine gröbere Abschätzung (4.32) der Grenzwerte für unterschiedliche Werte von $T_{\{<,>\}}$ dargestellt. Dabei ist $T_{\{<,>\}}$ die Anzahl der Schritte, die für den Ausstieg aus dem lokalen Extrempunkt benötigt wird. Je größer der Parameter $T_{\{<,>\}}$ gesetzt wird, desto weniger Aussagekraft hat die Abschätzung, desto konservativer sind die theoretischen Prognosen. Das zulässige Intervall für Parameter k wird für immer größere γ immer schmaler, siehe blaue Kurve für $T_{\{<,>\}} = 20$. Diese Prognose besagt, dass im Fall $\gamma \neq 1.0$ keine Einbindung der heuristischen Funktion empfohlen wird ($k \to 0$).
Die Werte für $T_{<,>}$ können auch geschätzt werden. Diese Werte können in Abhängigkeit von der gesamten Anzahl der Zustände im Zustandsraum bestimmt werden. An dieser Stelle kann $T_{\{<,>\}}$ durch den Wert $2 \cdot \sqrt{|S|}$ abgeschätzt werden, wobei $|S|$ die Anzahl der Elemente im Zustandsraum ist. Der Grund für diese Abschätzung ist die Vermutung, dass die Bewältigung der ungünstigsten Situation den Agenten einen maximalen Weg mit der Länge des halben Umfangs des Zustandsraumes kostet.
Wenn ein grobes Modell der Welt vorhanden ist, sollte die Abschätzung (4.20) angewendet werden. Dabei muss zuerst eine Prüfung der Existenz der Zykluskandidaten für den gewählten Diskontierungsfaktor γ durchgeführt werden, um diese Abhängigkeit der Grenzwerte vom Parameter $\widetilde{\Delta}(\gamma)$, $T_<$ und $T_>$ beizubehalten.

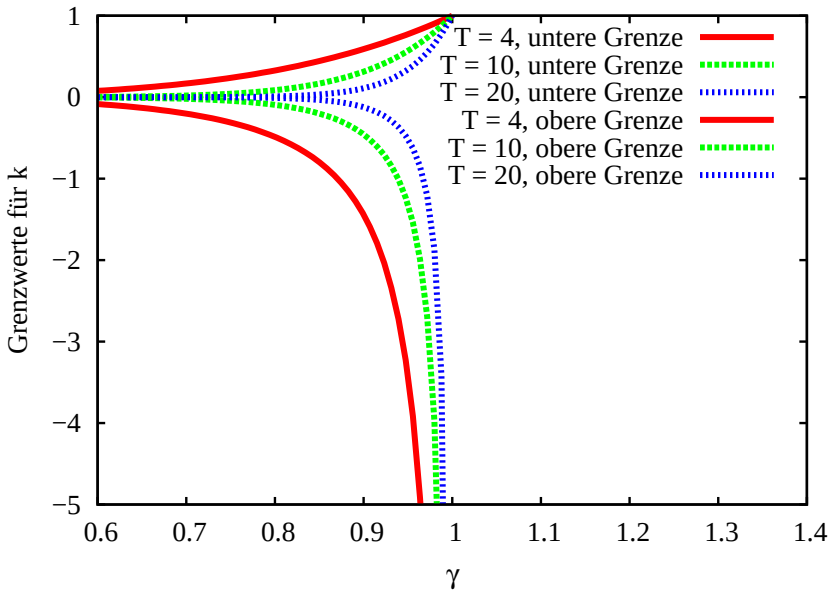


Abbildung 4.9: Grenzwerte für das zulässige Intervall für den Parameter k in Abhängigkeit vom Diskontierungsfaktor γ und der Anzahl der Schritte T, mit einer gröberen Abschätzung (4.32)

4.5 Ergebnisse in der Gitterwelt

An dieser Stelle werden die Ergebnisse aus dem Gitterweltproblem in zwei unterschiedlichen Szenarien präsentiert. Die kleine Anzahl an Zuständen im Zustandsraum und die vorhandene optimale Strategie erlauben genaue Aussagen für beide Szenarien über die erzielte Lerngeschwindigkeit und Lernqualität. Aus der Erhöhung der Lerngeschwindigkeit wird auf die Effizienzsteigerung der erweiterten Algorithmen geschlossen. Der Lernprozess wird in beiden Szenarien dann angehalten, wenn der angewendete Lernalgorithmus die optimale Strategie, die im Vorfeld bekannt ist, erreicht hat. Das Ziel bei der Ermittlung dieser Ergebnisse war es, sich einen ersten Überblick über die Möglichkeiten der heuristischen Erweiterung zu verschaffen. Die Ergebnisse für das eingeführte MCP und das Navigationsproblem stellen den allgemeineren Überblick über die Effizienzsteigerung der erweiterten Lernalgorithmen dar.
Die angewendeten Szenarien im Gitterweltproblem sind schon aus den theoretischen Überlegungen für das zulässige Intervall für den Parameter k bekannt. Es ist die Umgebung mit Weggabelung, siehe Abbildung 4.2(a), und die Umgebung mit Sackgassen, siehe Abbildung 4.3(a). Der Lernschritt aus Gleichung (3.10) bzw. der erweiterte Lernschritt aus Gleichung (4.2) werden für eine iterative Annäherung der optimalen Q-Funktion angewendet. Dieser Adaptionsschritt wird aber anders gemacht als im Standard-SARSA-Algorithmus. In jedem Schritt, der auch als eine Episode bezeichnet wird, werden alle Q-Werte für alle Zustände der Reihe nach aktualisiert. Diese Vorgehensweise ist wegen der kleinen Anzahl der Zustände im Zustandsraum möglich und stammt aus [Sutton & Barto, 1998, Kap. 4.2, S. 94]. Nach jeder Episode wird geprüft, ob die durch die aktuelle Q-Funktion repräsentierte Strategie mit der vorhandenen optimalen Strategie übereinstimmt. Der Schritt der Bildung der Strategie wird entweder mit einer ϵ-gierigen bzw. ϵ^*-gierigen Strategie durchgeführt. Der zufällige Anteil ϵ wird aber zu Null gesetzt. Wenn die optimale Strategie erreicht wird, werden noch weitere Aktualisierungsschritte durchgeführt, um die Stabilität der erzielten Lösung (optimale Strategie) zu prüfen. Diese zusätzlichen Schritte werden aber nicht zu den Ergebnissen hinzugezählt.

4.5.1 ϵ^*-gierige Strategie

Die Auswirkung der Erweiterung durch eine ϵ^*-gierige Strategie auf die Lerngeschwindigkeit wird für das Gitterweltproblem nicht festgehalten, da sich die hier angewendete Vorgehensweise für die Aktualisierung der Q-Funktion von der Anwendung der ϵ-gierigen Strategie zur Bestimmung jeder nächsten Aktion, wie es im Standard-SARSA-Algorithmus vorgesehen ist, unterscheidet.
Für die angewendete Vorgehensweise zur Aktualisierung der Q-Funktion und für die anschließend angewendete gierige Wahl bzw. heuristisch erweiterte, gierige Wahl (als ϵ^*-gierige Strategie bezeichnet) zur Bildung der aktuellen Strategie wird eine Beschleunigung der Lerngeschwindigkeit im Vergleich zur Standard-Methode von einer Episode für die ϵ^*-gierige Strategie in beiden Szenarien gemessen.
Zum Beispiel werden in der Umgebung mit Weggabelung statt 8 Episoden 7 Episoden bis zum Erreichen der optimalen Strategie mit der besten Parametrisierung der heuristischen Funktion benötigt, was eine Beschleunigung des Lernprozesses um 12.5% darstellt.

4.5.2 SARSA*-Algorithmus

Die Auswirkung des SARSA*-Algorithmus auf die Lerngeschwindigkeit des Lernprozesses wird durch den Vergleich mit dem Standard-SARSA-Algorithmus ermittelt. Für das Gitterweltproblem wird der Lernprozess für unterschiedliche Ausprägungen des heuristischen Einflussparameters k und des Diskontierungsfaktors $\gamma \in \{1.0, 0.9, 0.8, 0.7, 0.6\}$ in zwei Szenarien (Umgebung mit Weggabelung und Umgebung mit Sackgassen) durchgeführt. Die heuristische Funktion ist als normierte, parametrisierte Abstandsfunktion bis zum Zielzustand definiert. Der Adaptionsschritt wird durch die Update-Regel (4.2) für SARSA* bzw. die Update-Regel (3.10) realisiert. Die Anzahl der benötigten Episoden[28] bis zum Erreichen der optimalen Strategie wird festgehalten.
In Abbildung 4.10 ist die Abhängigkeit der Anzahl der bis zur optimalen Strategie benötigten Episoden vom heuristischen Einflussparameter k dargestellt. Jede Kurve in der Abbildung repräsentiert diese Abhängigkeit für eine Ausprägung des Diskontierungsfaktors γ. Die rote Linie kennzeichnet die bis zum Erreichen der optimalen Strategie benötigte Anzahl an Episoden ohne Einbindung der heuristischen Funktion im Lernprozess ($k = 0$). Alle Punkte, die unterhalb der roten Linie liegen, weisen eine Beschleunigung der Lerngeschwindigkeit auf.
Anhand dieser Abbildung kann die im oberen Kapitel theoretisch hergeleitete Abhängigkeit der Breite des zulässigen Intervalls für den Parameter k vom Diskontierungsfaktor $0 < \gamma \leq 1.0$ beobachtet werden. Für $\gamma = 0.6$ ist die gelbe Parabel sehr schmal. Nur die Einbindung der heuristischen Funktion mit den nahe an Null liegenden Werten für den heuristischen Einflussparameter k in den Lernprozess konvergiert zur gesuchten optimalen Strategie. Für die grüne Kurve dagegen, die den Lernprozess mit $\gamma = 1.0$ darstellt, erreicht jede abgebildete Ausprägung des heuristischen Einflussparameters $k \in [-6, 1.0]$ nach einer endlichen Anzahl an Lernschritten die gesuchte optimale Strategie.
Ein richtig gewählter Parameter k weist eine Beschleunigung der Lerngeschwindigkeit von bis zu 50% auf. Um diesen Effekt besser beobachten zu können wird in Abbildung 4.11 ein Ausschnitt der Abbildung 4.10 vergrößert dargestellt. Eine Beschleunigung der Lerngeschwindigkeit wird für jede Wahl des Diskontierungsfaktors γ und des heuristischen Einflussparameters $k \in [0.1, 0.4]$ gemessen. Statt acht Episoden bis zum Erzielen der optimalen Strategie werden im besten Fall bei Anwendung des SARSA*-Algorithmus nur vier Episoden benötigt.
Die durch eine heuristische Funktion mit dem optimalen Wert des Parameters $k > 0$ repräsentierte

[28] Eine Episode beinhaltet einen Adaptionsschritt der Q-Funktion, in dem alle Q-Werte für alle Zustände der Reihe nach aktualisiert werden.

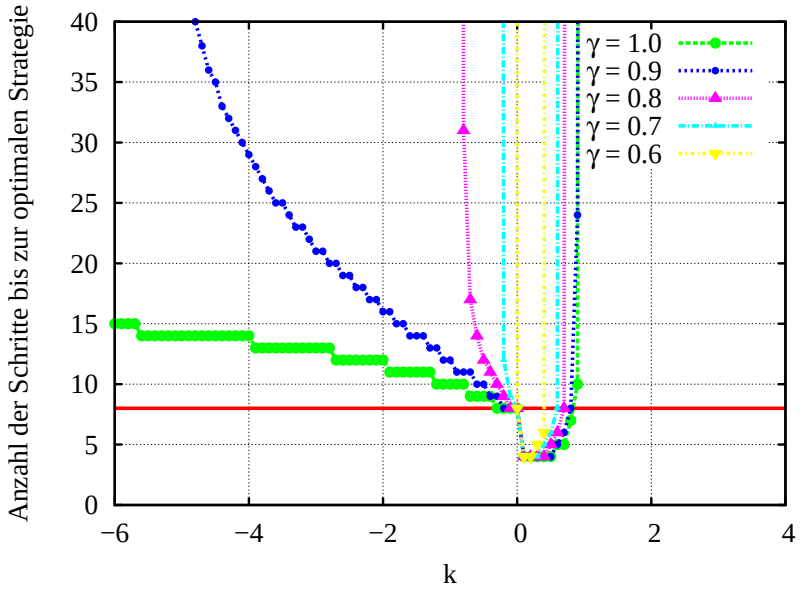


Abbildung 4.10: Abhängigkeit der Anzahl der bis zur optimalen Strategie benötigten Episoden, siehe Gleichung (4.2), vom heuristischen Einflussparameter k und Diskontierungsfaktor γ in der Umgebung mit Weggabelung. Die rote Linie stellt die Anzahl der bis zur optimalen Strategie benötigten Episoden ohne Heuristik dar.

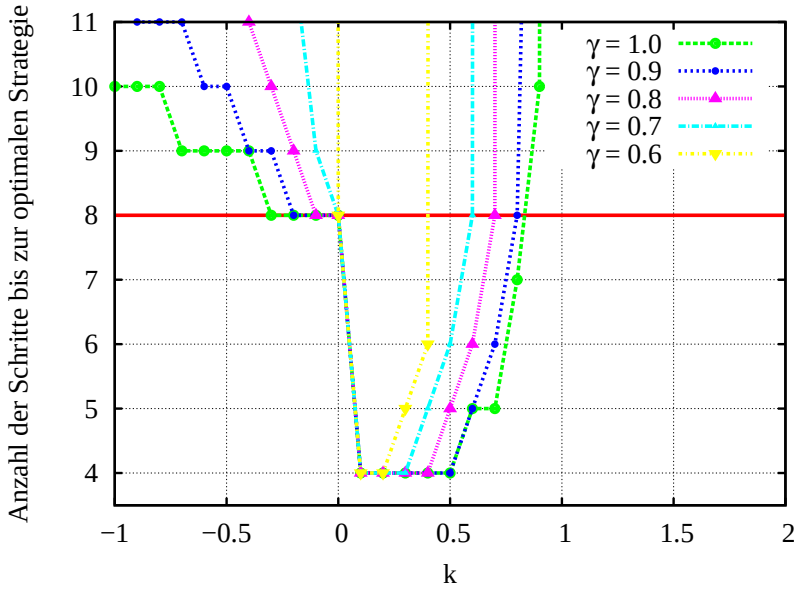


Abbildung 4.11: Vergrößerter Ausschnitt aus Abbildung 4.10. Die Anzahl der benötigten Episoden bis zum Erreichen der optimalen Strategie für die Lernmethode mit heuristischem Einfluss und richtiger Parametrisierung ist deutlich niedriger als die Anzahl der Episoden ohne heuristischen Einfluss. Zum Beispiel für den Parametersatz $\gamma = 1.0$ und $k \in [0.1, 0.4]$ liegt die Anzahl der Episoden bei 4 statt 8.

Heuristik steht in Konflikt zur Problemstellung. Die Aufgabe des Agenten lautet, den Zielzustand s_0 zu erreichen. Die verfolgte Heuristik besagt, dass das Erreichen des am weitesten vom Zielzustand s_0 liegenden Zustands s_{11} den größten Nutzen bringt. Der Zustand s_{11} ist aber ein wichtiger Punkt, der

auf dem optimalen Weg zum Ziel liegt. Die heuristische Funktion wirkt sich positiv beim Lernen des optimalen Abschnitts des Weges bis zum Zustand s_{11} aus. Mithilfe des SARSA-Ansatzes wird dann der restliche Abschnitt vom Zustand s_{11} erlernt und die negative Auswirkung der heuristischen Funktion an diesem Wegabschnitt schnell beseitigt.
Die gewählten Umgebungen sind so konstruiert, dass die heuristische Funktion für einen Teil des Zustandsraumes in Widerspruch zur optimalen Strategie liegt. Die Umgebung mit Weggabelung ist ziemlich speziell. Die Umgebung mit Sackgassen stellt ein gewöhnlicheres Szenario in der Gitterwelt dar.
In der Umgebung mit Sackgassen benötigt die heuristische Funktion keine so feine Justierung des Pa-

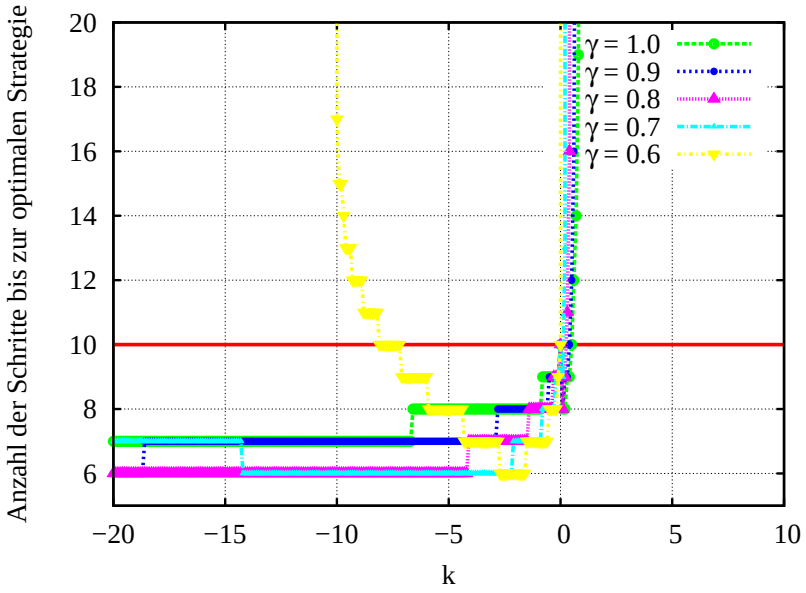


Abbildung 4.12: Abhängigkeit der Anzahl der bis zur optimalen Strategie benötigten Episoden, siehe Gleichung (4.2), vom heuristischen Einflussparameter k für unterschiedliche Diskontierungsfaktoren γ in der Umgebung mit Sackgassen. Die rote Linie stellt die Anzahl der bis zur optimalen Strategie benötigten Episoden ohne Heuristik dar.

rameters k wie im vorherigen Szenario für die Umgebung mit Weggabelung. Wenn der Parameter γ für den Parameter $k \leq -4$ entsprechend groß gewählt wird ($\gamma \geq 0.7$), wird immer eine Beschleunigung der Lerngeschwindigkeit beobachtet. Alle Kurven für $\gamma = \{1.0, 0.9, 0.8, 0.7\}$ liegen für $k < 0$ unterhalb der roten Linie, siehe Abbildung 4.12. Für den Parametersatz $\gamma \geq 0.7$ und $k < 0$ werden bis zum Erreichen der optimalen Strategie anstatt der 10 Episoden mit dem SARSA-Algorithmus nur 6 bzw. 7 Episoden mit dem SARSA*-Algorithmus benötigt. Dies bedeutet eine Beschleunigung von 40%. Der große Vorteil dabei liegt in einem breiten zulässigen Intervall für den heuristischen Einflussparameter k, in dem die Beschleunigung der Lerngeschwindigkeit erzielt wurde.
Der Reinforcement-Learning-Ansatz wird durch die Einbindung der richtig parametrisierten heuristischen Funktion im Gitterweltproblem um 40% bis 50% effizienter. Der Erfolg der Anwendung des SARSA*-Algorithmus hängt von der richtigen Kalibrierung des heuristischen Einflussparameters ab.

4.6 Ergebnisse für das Mountain-Car-Problem

Das Mountain-Car-Problem (MCP), siehe Kapitel 2.3, ist für das nicht endliche Markow-Entscheidungsproblem (MDP) definiert und ist ein Benchmark-Problem für Reinforcement-Learning-Methoden. Aus

diesem Grund wird die in diesem Kapitel vorgeschlagene Erweiterung des Reinforcement-Learning-Ansatzes um die heuristische Funktion hinsichtlich des MCP analysiert. Ein Teil der hier beschriebenen Ergebnisse ist bereits von der Autorin in [Noglik et al., 2009] publiziert worden.
Die Definition des Zustandsraumes und die Bewegung des Agenten wurde in Kapitel 2.3.1 eingeführt. Der Agent hat die Startkoordinate $x^{Start} = -0.523$ und die Zielkoordinate $x^{Ziel} = 0.6$. In diesem Abschnitt werden folgende Einstellungen für das MCP angewendet. Die Parameter werden zu $\alpha = 0.2$, $\epsilon = 0$ berechnet. Für die Zerlegung des Zustandsraums wird die Tile-Coding-Methode mit einer Zerlegung von 11×11 Teilen und einem Tiling angewendet. Der Parameter λ, der Diskontierungsfaktor γ und der heuristische Einflussparameter k werden variiert.

4.6.1 Definition der heuristischen Funktion

Die im MCP angewendete heuristische Funktion repräsentiert die normierte Abstandsfunktion bis zum vorgegebenen Zustand und beinhaltet nur die Positionskomponente des Zustandsraumes. Die Geschwindigkeitskomponente des Agenten wird in der heuristischen Funktion nicht berücksichtigt.
Wenn der vorgegebene Zustand an den Zielzustand gesetzt wird, entsteht die heuristische Standard-Funktion, die schon im oberen Kapitel eingeführt worden ist, siehe Gleichung (4.33). Die verfolgten Heuristiken unterscheiden sich je nach Vorzeichen des heuristischen Einflussparameters k. Für $k > 0$ wird die folgende Heuristik verfolgt: "je weiter sich der Agent von der Zielposition befindet, desto besser ist seine Lage". Für $k < 0$ lautet die Heuristik "je näher der Agent am Ziel liegt, desto besser ist seine Lage". Die heuristische Standard-Funktion (4.33) wird im SARSA*-Algorithmus angewendet.

$$h(x) = k \cdot \frac{\sqrt{(x - \underbrace{0.6}_{:=x^{Ziel}})^2}}{\underbrace{1.8}_{:=|0.6|+|1.2|}} \tag{4.33}$$

Die erzielten Ergebnisse bei der Anwendung der heuristischen Standard-Funktion in der ϵ^*-gierigen Strategie waren nicht aussagekräftig. Aus diesem Grund wurde eine neue heuristische Funktion eingeführt, die die grundlegende Idee der Einbindung der Heuristik in der Aktionswahl besser widerspiegelte.
Eine Alternative zur Definition der heuristischen Standard-Funktion basiert auf der Startposition des Agenten, siehe Gleichung (4.34). Die alternative heuristische Funktion (4.34) ist auch normiert, so dass der Wertebereich für diese Funktion im Intervall $[0, 1]$ liegt. Der Agent startet an der tiefsten Stelle im Tal, in der Position $x^{Start} = -0.523$. Die neu aufgestellte heuristische Funktion für positive Parameter $k > 0$ gibt den Zuständen eine bessere Bewertung, die sich weiter weg vom Startzustand befinden. Die verfolgte Heuristik lautet dabei "Je weiter der Zustand von der Startposition entfernt liegt, desto besser ist die Lage des Agenten". Für $k < 0$ besagt die verfolgte Heuristik, "je näher der Agent an der Startposition liegt, desto besser ist seine Lage". Mit einer so definierten heuristischen Funktion, angewendet für die Bildung der ϵ^*-gierigen Strategie und mit dem Lernschritt des Standard-SARSA-Algorithmus wurden aussagekräftige Ergebnisse erzielt.

$$h(x) = k \cdot \frac{\sqrt{(x - \underbrace{-0.523}_{:=x^{Start}})^2}}{1.8} \tag{4.34}$$

An dieser Stelle wird die Motivation zur neuen Definition der heuristischen Funktion, die in der Aktionswahl angewendet wird, erläutert. Für die unterschiedlichen Aktionen ($a = 1$: beschleunige nach vorne, $a = 0$: beschleunige nicht, $a = -1$: beschleunige nach hinten) gilt $x_{t+1}^{a=1} \geq x_{t+1}^{a=0} \geq x_{t+1}^{a=-1}$,

wobei die Bezeichnung $x_{t+1}^{a=1} := \tau(x_t, a = 1)$ den Übergang vom Zustand x_t in den Zustand x_{t+1} bei Anwendung der Aktion a bedeutet. Die Veränderung der Position und der Geschwindigkeit in Abhängigkeit von der durchgeführten Aktion ist im Gleichungssystem (2.1), (2.2) dargestellt. Für $k > 0$ und eine als Distanz bis zum weitesten Punkt $x^{Ziel} = 0.6$ definierte heuristische Funktion, siehe Gleichung (4.33), gilt $h(x_{t+1}^{a=-1}) \geq h(x_{t+1}^{a=0}) \geq h(x_{t+1}^{a=1})$. Für $k < 0$ und eine als Distanz bis zum Zielpunkt definierte heuristische Funktion gilt: $h(x_{t+1}^{a=-1}) \leq h(x_{t+1}^{a=0}) \leq h(x_{t+1}^{a=1})$. Die heuristische Funktion gilt stets für eine Aktion, entweder beschleunige rückwärts für $k < 0$ oder beschleunige vorwärts für $k > 0$. Der Vorschlag des heuristischen Anteils ist nicht immer richtig. Für den Fall, dass das Fahrzeug genügend Schwung geholt hat, ist der Vorschlag der heuristischen Funktion weiter vorwärts zu beschleunigen ($k > 0$) richtig. Ebenso, wenn mehr Schwung geholt werden soll, ist der Vorschlag der heuristischen Funktion weiter rückwärts zu beschleunigen ($k < 0$) richtig. Hingegen ist der Vorschlag der heuristischen Funktion falsch, wenn im ersten Fall der Agent noch nicht weit genug auf dem rechten Hang ist um das Ziel zu erreichen. Und im zweiten Fall, nachdem der Agent genug Schwung geholt hat und das Ziel erreichen könnte, aber seine Heuristik ihm empfiehlt rückwärts zu beschleunigen. Die zum Erzielen einer optimalen Strategie benötigten Aktionen widersprechen dann in beiden Fällen dem Vorschlag der heuristischen Funktion, so dass die Anwendung der heuristischen Standard-Funktion für die Bildung der ϵ^*-gierigen Strategie nicht die gewünschte aussagekräftige Heuristik widerspiegelt.

4.6.2 ϵ^*-gierige Strategie

Für die heuristische Funktion in der ϵ^*-gierigen Strategie werden Ergebnisse für beide Definitionen der heuristischen Funktion (Standard und Alternative) bestimmt.

4.6.2.1 Auf dem Zielpunkt basierende heuristische Funktion

Bei der Einbindung der heuristischen Standard-Funktion (4.33) in die ϵ^*-gierige Strategie wurde für beide Vorzeichen des heuristischen Einflussparameters sowohl eine Beschleunigung als auch eine Verlangsamung des Lernprozesses gemessen. Die durch eine heuristische Standard-Funktion definierte Heuristik, die zur Bildung der ϵ^*-gierigen Strategie verwendet wird, liefert keine eindeutige Aussage über die Güte der Zustände im Zustandsraum des MCP. Die im vorliegenden Unterkapitel erzielten Ergebnisse offenbaren diese Hypothese.

In den Abbildungen 4.13, 4.14 sowie 4.15 sind die relativen Werte des Messkriteriums Lernfortschritt (LP-Kriterium, vergleiche Unterkapitel 3.6) für $k \in \{-10, -5, -3, -2, -1,\ 1, 2, 3\}$ und $\gamma \in \{0.97, 0.98, 0.99, 1.0\}$ in Relation zur RL-Methode ohne heuristischen Einfluss dargestellt. Die drei Abbildungen unterscheiden sich in der Stärke der Auswirkung der Eligibility-Traces. In Abbildung 4.13 sind die Ergebnisse mit dem Parameter $\lambda = 0.9$ erzielt worden, in Abbildung 4.14 mit dem Parameter $\lambda = 0.95$, und in Abbildung 4.15 werden die Eligibility-Traces abgeschaltet ($\lambda = 0$). Auf der Grundlage der Veränderung des Lernfortschritts in % wird die Beschleunigung bzw. Verlangsamung des Lernprozesses gemessen. Die Punkte, die oberhalb des durch eine Gerade angedeuteten Nullniveaus liegen, deuten auf die Beschleunigung, also Verbesserung der Geschwindigkeit im Lernprozess hin. Die Punkte unterhalb des Nullniveaus deuten auf eine Verlangsamung, also Verschlechterung der Geschwindigkeit des Lernprozesses hin.

Auf Grundlage der Abbildungen 4.13, 4.14 sowie 4.15 kann keine allgemeine Tendenz, ob die heuristische Funktion einen echten Nutzen bringt, erkannt werden. Die Punkte liegen sowohl oberhalb als auch unterhalb des Nullniveaus in allen drei Abbildungen. Unabhängig von den Vorzeichen des heuristischen Einflussparameters k beträgt die erzielte Beschleunigung des Lernprozesses fast in allen Fällen

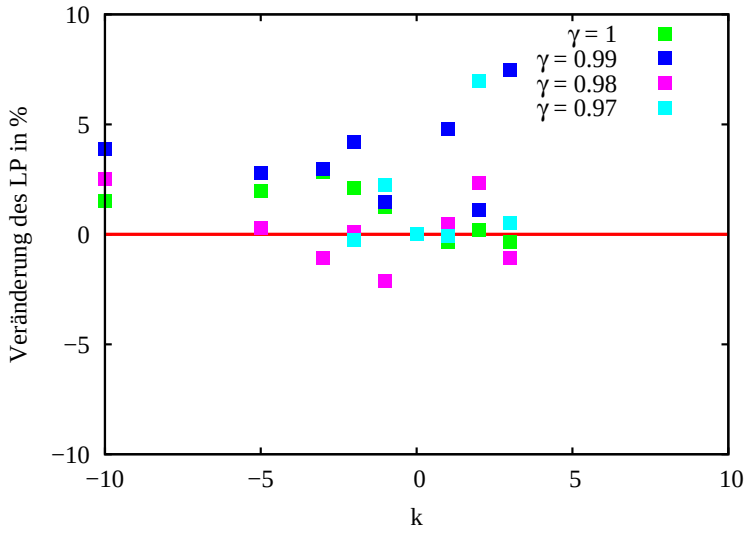


Abbildung 4.13: Veränderung des Lernfortschritts im MCP mit heuristischer Standard-Funktion (4.33) in Abhängigkeit vom Parameter k für verschiedene γ-Parameter bei $\lambda = 0.9$

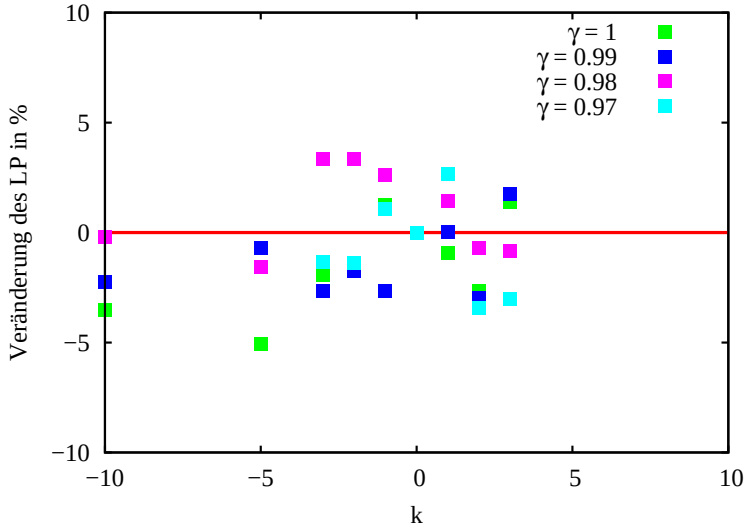


Abbildung 4.14: Veränderung des Lernfortschritts im MCP mit heuristischer Standard-Funktion (4.33) in Abhängigkeit vom Parameter k für verschiedene γ-Parameter bei $\lambda = 0.95$

nicht mehr als 5%, siehe Abbildungen 4.13. Es liegen Fälle vor, in denen die Verlangsamung des Lernprozesses bis zu -5% beträgt, siehe Abbildung 4.14. Für fast alle Ergebnisse liegt kein signifikanter Unterschied zur Standard-Methode ohne heuristischen Einfluss vor. Nur für zwei Fälle wird eine signifikante Beschleunigung des Lernprozesses im Vergleich zur Standard-Methode gemessen. Für $\lambda = 0.9$, ($\gamma = 0.99$, $k = 3$) und ($\gamma = 0.97$, $k = 2$) liegt die Beschleunigung des Lernprozesses bei 7%, siehe Abbildungen 4.13.

Die Erweiterung des RL-Algorithmus um die ϵ^*-gierige Strategie in Verbindung mit der heuristischen Funktion (4.33) ist für das Mountain-Car-Problem nicht empfehlenswert, da keine eindeutige Verbesserung erzielt worden ist. Es konnte keine Tendenz für die Bestimmung des optimalen heuristischen Einflussparameters k erkannt werden. Auf Grund der erzielten Ergebnisse hat die oben aufgestellte The-

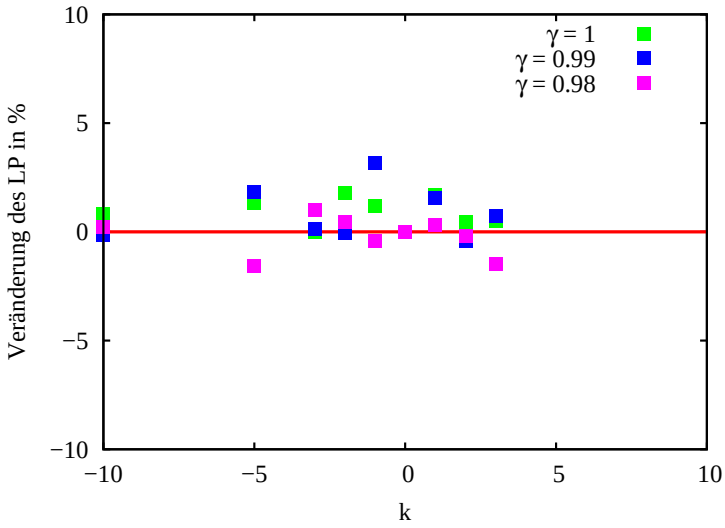


Abbildung 4.15: Veränderung des Lernfortschritts im MCP mit ϵ^*-gieriger Strategie in Abhängigkeit vom Parameter k für verschiedene γ-Parameter bei $\lambda = 0$

se über die nicht genügende Aussagekraft der heuristischen Standard-Funktion für die Bestimmung der aktuellen Aktion mit der ϵ^*-gierigen Strategie ihre Bestätigung gefunden.

4.6.2.2 Auf dem Startpunkt basierende heuristische Funktion

Für die in Gleichung (4.34) definierte heuristische Funktion und $k > 0$ wird die Heuristik verfolgt, dass "je weiter der Agent vom Startpunkt entfernt ist, desto besser sein Zustand ist". Die so aufgestellte Heuristik stimmt mit der Aufgabenstellung überein.
In Abbildung 4.16 sind die relativen Werte des Messkriteriums Lernfortschritt im Vergleich zur Methode ohne heuristischen Einfluss dargestellt. Die Parameter $k \in \{-2, -1, 1, 2, 3, 5, 10\}$ und $\gamma \in \{0.97, 0.98, 0.99, 1.0\}$ werden variiert. Der Parameter $\lambda = 0.95$ wird festgehalten.
Aus Abbildung 4.16 wird ersichtlich, dass für die Werte $k > 0$ eine stabile Beschleunigung des Lernprozesses um bis zu 5% durch Anwendung der ϵ^*-gierigen Strategie für $k = 10$ und $\gamma \in \{0.97, 0.98, 0.99\}$ erzielt wurde. Die mit dem Parameterwert $\gamma = 1.0$ erzielten Ergebnisse sind unerwartet schlecht.
Für eine bessere Nachvollziehbarkeit werden einige Ergebnisse aus dieser Abbildung in Form von zwei Tabellen dargestellt. In Tabelle 4.1 werden die Ergebnisse für $\gamma = 0.97$ und $k \in \{1, 2, 3, 5, 10\}$ und in Tabelle 4.2 werden die Ergebnisse für $\gamma = 0.99$ und $k \in \{-0.2, -0.2, 1, 2, 3, 10\}$ dargestellt. Die erste Spalte beider Tabellen stellt unterschiedliche Ausprägungen des heuristischen Einflussparameters k dar. In der zweiten Spalte beider Tabellen wird der relative Wert des LP-Kriteriums in % abgebildet. Die dritte Spalte gibt an, ob der erzielte Unterschied zur RL-Methode ohne heuristischen Einfluss signifikant ist. Die Aussagekraft der erzielten Ergebnisse wird mithilfe des t-Tests kontrolliert, siehe Kapitel 3.6. Es ist eine binäre Aussage: 0 bedeutet "Ergebnis ist nicht aussagekräftig", 1 bedeutet "Aussagekraft wurde bestätigt". Die Spalte mit der Aussagekraft der Ergebnisse ist mit "t-Test LP" überschrieben.
Eine signifikante Verbesserung für $k = 10$ von bis zu 5.21%, siehe Tabelle 4.1, und bis zu 4.44%, siehe Tabelle 4.2, wird gemessen.
Wenn die heuristische Funktion aus Gleichung (4.34) mit negativem heuristischen Einfluss $k < 0$ zur Bestimmung der nächsten Aktion einbezogen wird, steht die so definierte heuristische Funktion in Konflikt zur Aufgabenstellung. Die Heuristik besagt in diesem Fall, dass ein Verhalten des Agenten gut ist, wenn

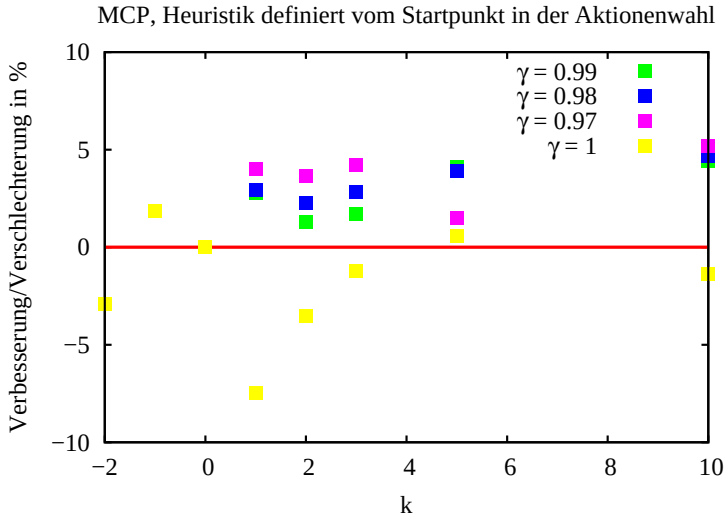


Abbildung 4.16: Veränderung des Lernfortschritts im MCP mit einer auf dem Startpunkt basierenden heuristischen Funktion und ϵ^*-gieriger Strategie in Abhängigkeit vom Parameter k für verschiedene γ-Parameter bei $\lambda = 0.95$

Tabelle 4.1: Relative Veränderung des Lernfortschritts bei Anwendung der ϵ^*-gierigen Strategie und des Standard-SARSA-Algorithmus im MCP im Vergleich zur Standard-Methode. Parametrisierung: $\gamma = 0.97, \lambda = 0.95$.

k	LP	t-Test LP
10	5.21%	1
5	1.47%	0
3	4.19%	1
2	3.63%	1
1	3.99%	1

Tabelle 4.2: Relative Veränderung des Lernfortschritts bei Anwendung der ϵ^*-gierigen Strategie und des Standard-SARSA-Algorithmus im MCP im Vergleich zur Standard-Methode. Parametrisierung: $\gamma = 0.99, \lambda = 0.95$.

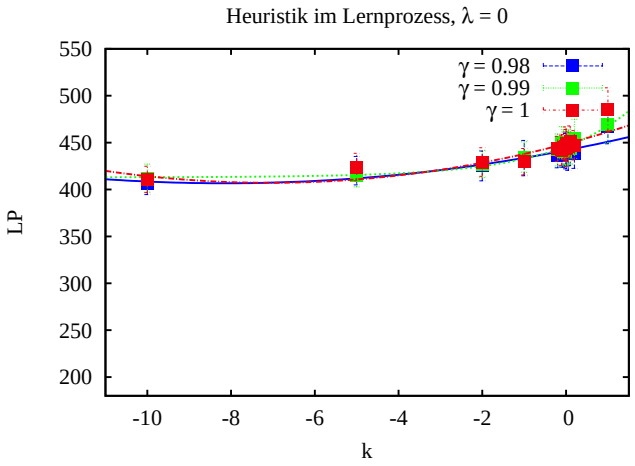
k	LP	t-Test LP
10	4.44%	1
5	4.13%	1
3	1.72%	0
2	1.28%	0
1	2.78%	0
-0.1	4.29%	1
-0.2	-180.56%	1

der Agent sich nahe am oder gar im Startpunkt befindet. Die Ergebnisse für unterschiedliche $\gamma < 1.0$ für $k \leq -0.2$ weisen eine Divergenz des Verfahrens für die erweiterte Methode auf, siehe letzte Zeile in Tabelle 4.2. Wenn der heuristische Einfluss zu stark gewichtet wird, ist es aus Sicht des Agenten viel einträglicher im Tal zu verbleiben und für jeden Schritt bestraft zu werden, anstatt herauszufahren. Die

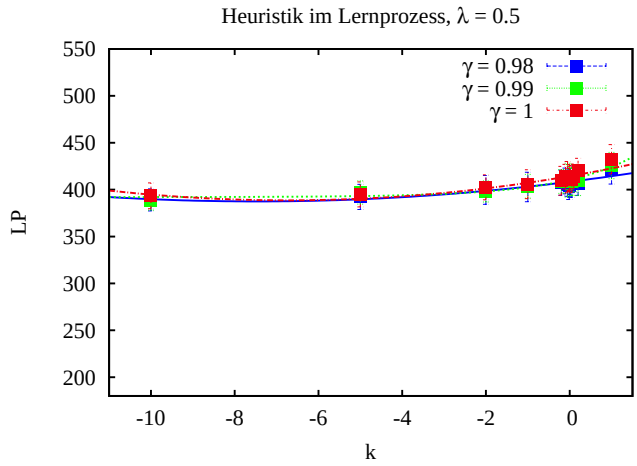
heuristische Funktion schlägt immer die Aktion ”nichts tun” als die beste Variante vor. Auf der Grundlage der aufgestellten Ungleichung (4.7) wird das zulässige Intervall für den Parameter k aufgestellt. Da die heuristische Funktion für $k < 0$ der Problemstellung am stärksten im Startpunkt $x^{Start} = -0.523$ widerspricht, und aus der optimalen Lösung bekannt ist, dass ungefähr 150 Schritte bis zum Erreichen des Ziels, also bis zum Ausstieg aus dem Tal, bei Anwendung der optimalen Strategie benötigt werden, wird das zulässige Intervall des Parameters k wie folgt abgeschätzt: $-\gamma^{150} \cdot \frac{1,8}{2} \leq k \leq 0$. Somit ist für $\gamma = 0.97$ das zulässige Intervall für den heuristischen Einflussparameter $k \in [-0.009, 0]$, und für $\gamma = 0.99$ ist das zulässige Intervall $k \in (-0.2, 0]$.

4.6.3 SARSA*-Algorithmus

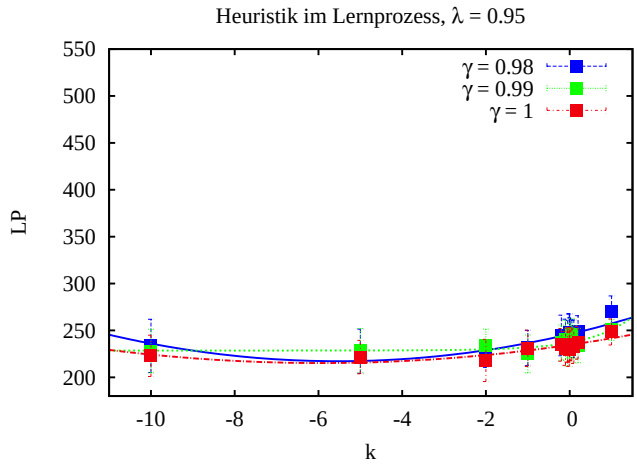
In diesem Unterkapitel wird die erzielte Lerngeschwindigkeit mit dem SARSA*-Algorithmus mit der erzielten Lerngeschwindigkeit mit gewöhnlichen SARSA-Algorithmus verglichen. Das Lernen mit dem SARSA*-Algorithmus erfolgt mit der heuristischen Funktion (4.33) und der erweiterten Lernregel (4.2). Die heuristische Standard-Funktion beinhaltet die richtige Tendenz, je näher der Agent am Ziel ist, desto besser ist seine Lage. Durch die Einbindung der heuristischen Funktion in der Q-Funktion werden die kompletten Ketten aus Entscheidungen mitberücksichtigt, so dass die von der heuristischen Standard-Funktion ermittelte richtige Tendenz besser zur Geltung kommt als die Berücksichtigung von nur einer Aktion für die Bewertung eines Zustands, wie es bei der ϵ^*-gierigen Strategie ist.
In Abbildung 4.17 sind die absoluten Werte für die Lerngeschwindigkeit (Lernfortschritt) für die um eine heuristische Funktion im Lernprozess erweiterte Lernmethode in Abhängigkeit vom Parameter k für unterschiedliche Diskontierungsfaktoren $\gamma \in \{0.98, 0.99, 1.0\}$ und eine unterschiedliche Stärke der Ausprägungen der Eligibility-Traces $\lambda \in \{0, 0.5, 0.95\}$ aufgetragen. Dabei ist die Varianz der jeweiligen Werte durch vertikale Balken gekennzeichnet. Die große Auswirkung des Parameters λ auf die Lerngeschwindigkeit (Lernfortschritt) wird anhand der absoluten Werte veranschaulicht. Die drei über die ermittelten Punkte geschätzten Kurven liegen auf unterschiedlichen Niveaus. Der Lernprozess ist am schnellsten für die stark ausgeprägten Eligibility-Traces ($\lambda = 0.95$). Das erreichte Niveau der beiden anderen Kurven unterscheidet sich kaum.
Der positive Einfluss der Erweiterung kann anhand der dargestellten Abbildungen nur vermutet werden. Um diese Vermutung besser zu erfassen und die Ergebnisse besser zu vergleichen, sind die relativen Werte des Lernfortschritts in den Tabellen 4.3, 4.4, 4.5 eingetragen. Jeder Wert für LP entspricht der relativen Veränderung der Lerngeschwindigkeit im Vergleich zur Standard-Methode ohne heuristischen Einfluss ($k = 0$). Dieser Wert besagt, ob die Einbindung der heuristischen Funktion in die Lernregel den Lernprozess beschleunigt hat. Der heuristische Einflussparameter k und der angewendete Wert des Diskontierungsfaktors γ werden variiert und sind mit ”LP” und γ überschrieben. Die Tabellen unterscheiden sich in der Stärke der Einbindung der Eligibility-Traces voneinander.
In Tabelle 4.3 sind die Ergebnisse des Lernprozesses für die Parameter $\lambda = 0$ und $\gamma \in \{1.0, 0.99, 0.98\}$ eingetragen. Eine signifikante Verschlechterung wird für den Parameterwert $k = 1.0$ für alle drei untersuchten Werte des Parameters γ gemessen, siehe 1. Zeile der Tabelle. Für den Parameterwert $k = -10$ wird für alle drei Parameterwerte von $\gamma \in \{1.0, 0.99, 0.98\}$ eine signifikante Beschleunigung des Lernprozesses um 8% gemessen, siehe letzte Zeile der Tabelle. Für $k \leq -2$ wird für jede Ausprägung des Diskontierungsfaktors γ eine signifikante Beschleunigung des Lernprozesses gemessen. Die Werte des heuristischen Einflussparameters $k \in [-0.2, 0.2]$ weisen keine aussagekräftige Auswirkung auf den Lernprozess auf. Diese Werte des heuristischen Einflussparameters sind nicht stark genug, um den Lernprozess im MCP zu beeinflussen.
In Tabelle 4.4 sind die Ergebnisse des Lernprozesses für die Parameter $\lambda = 0.5$ und $\gamma \in \{1.0, 0.99, 0.98\}$



(a) $\lambda = 0.0$



(b) $\lambda = 0.5$



(c) $\lambda = 0.95$

Abbildung 4.17: Absolute Werte der Lerngeschwindigkeit in Abhängigkeit vom Parameter k für das MCP und den SARSA*-Algorithmus, $\lambda = \{0, 0.5, 0.95\}$, $\epsilon = 0.0$, $\alpha = 0.1$

eingetragen. Es wird die gleiche Beobachtung gemacht wie für $\lambda = 0$. Es wird eine signifikante Verschlechterung für den Parameterwert $k = 1.0$ für alle drei untersuchten Werte des Parameters γ gemessen, siehe 1. Zeile der Tabelle. Dabei wird für unterschiedliche Parameterkombinationen $k = 0.2, \gamma =$

Tabelle 4.3: Relative Veränderung des durchschnittlichen Lernfortschritts im MCP bei Anwendung der Standard-Heuristik im Lernprozess und des SARSA*-Algorithmus im Vergleich zur Methode ohne Erweiterung für unterschiedliche Diskontierungsfaktoren $\gamma \in \{1.0, 0.99, 0.98\}$ bei $\lambda = 0$ und unterschiedliche Werte für den heuristischen Einflussparameter k

	$\gamma = 1.0$		$\gamma = 0.99$		$\gamma = 0.98$	
k	LP	t-Test LP	LP	t-Test LP	LP	t-Test LP
1	-8.04%	1	-4.53%	1	-6.02%	1
0.2	0.44%	0	-1.14%	0	0.5%	0
0.1	-0.42%	0	1.25%	0	0.04%	0
0.05	0.82%	0	0.78%	0	0.36%	0
0.01	1.09%	0	1.03%	0	-0.27%	0
-0.01	0.8%	0	1.8%	1	0.07%	0
-0.05	0.87%	0	1.88%	1	0.5%	0
-0.1	1.61%	1	-0.23%	0	0.78%	0
-0.2	1.31%	0	1.47%	1	0.95%	0
-1	4.34%	1	3.17%	1	1.67%	0
-2	4.43%	1	4.83%	1	3.57%	1
-5	5.6%	1	7.48%	1	4.71%	1
-10	8.56%	1	8.17%	1	7.84%	1

Tabelle 4.4: Relative Veränderung des durchschnittlichen Lernfortschritts im MCP bei Anwendung der Standard-Heuristik im Lernprozess und des SARSA*-Algorithmus im Vergleich zur Methode ohne Erweiterung für unterschiedliche Diskontierungsfaktoren $\gamma \in \{1.0, 0.99, 0.98\}$ bei $\lambda = 0.5$ und unterschiedliche Werte für den heuristischen Einflussparameter k

	$\gamma = 1.0$		$\gamma = 0.99$		$\gamma = 0.98$	
k	LP	t-Test LP	LP	t-Test LP	LP	t-Test LP
1	-4.9%	1	-2.92%	1	-4.2%	1
0.2	-1.98%	1	0.86%	0	-0.57%	0
0.1	-0.49%	0	0.7%	0	-1.59%	1
0.05	0.25%	0	1.21%	0	-0.61%	0
0.01	-0.1%	0	1.28%	0	-1.05%	0
-0.01	0.43%	0	0.05%	0	-0.17%	0
-0.05	-0.28%	0	0.85%	0	-0.14%	0
-0.1	-0.38%	0	1.17%	0	-1.14%	0
-0.2	0.65%	0	1.19%	0	-0.68%	0
-1	1.54%	1	2.42%	1	0.43%	0
-2	2.35%	1	3.76%	1	1.21%	0
-5	4.17%	1	4.02%	1	3.06%	1
-10	4.54%	1	6.02%	1	3.83%	1

1.0 und $k = 0.1, \gamma = 0.98$ auch eine signifikante Verschlechterung gemessen. Für den Parameter $k \leq -5$ wird für alle drei Werte von $\gamma \in \{1.0, 0.99, 0.98\}$ eine signifikante Beschleunigung des Lernprozesses von $3\% - 6\%$ gemessen, siehe letzte zwei Zeilen der Tabelle. Für die untersuchte Parameterkonfiguration beeinflusst die heuristische Funktion den Lernprozess für die untersuchten Werte des Parameters γ kaum, wenn der heuristische Einflussparameter k im Intervall $[-0.2, 0.05]$ liegt.
In Tabelle 4.5 sind die Ergebnisse für die Parameter $\lambda = 0.95$ und $\gamma \in \{1.0, 0.99, 0.98\}$ aufgetragen.

Tabelle 4.5: Relative Veränderung des durchschnittlichen Lernfortschritts im MCP bei Anwendung der Standard-Heuristik im Lernprozess und des SARSA*-Algorithmus im Vergleich zur Methode ohne Erweiterung für unterschiedliche Diskontierungsfaktoren $\gamma \in \{1.0, 0.99, 0.98\}$ bei $\lambda = 0.95$ und unterschiedliche Werte für den heuristischen Einflussparameter k

$\gamma = 1.0$			$\gamma = 0.99$		$\gamma = 0.98$	
k	LP	t-Test LP	LP	t-Test LP	LP	t-Test LP
1	-6.48%	1	-4.59%	1	-8.86%	1
0.2	-1.58%	0	2.4%	0	-0.54%	0
0.1	0%	0	0.97%	0	2.67%	0
0.05	-0.97%	0	-2.42%	0	1.4%	0
0.01	-0.79%	0	1%	0	1.44%	0
-0.01	1.44%	0	1.92%	0	0.43%	0
-0.05	-0.5%	0	3.24%	1	1.79%	0
-0.1	1.02%	0	-0.17%	0	1.78%	0
-0.2	-0.6%	0	1.17%	0	1.26%	0
-1	0.98%	0	6.11%	1	6.64%	1
-2	6.51%	1	2.27%	0	9.66%	1
-5	5%	1	4.65%	1	8.04%	1
-10	4.41%	1	4.84%	1	5.74%	1

Auch für diese Konfiguration der Parameter wird eine Verschlechterung des Lernfortschritts für einen falsch bestimmten Parameter von $k = 1.0$ beobachtet. Die höchste Beschleunigung von 9.66% wurde für das Parameterset ($\gamma = 0.98$, $k = -2$) erzielt. Der beste Wert für den heuristischen Einflussparameter liegt für die Werte $k \in [-1, 5]$ vor. Dieses Ergebnis unterscheidet sich vom Ergebnis aus den oberen Tabellen in denen der beste Wert des heuristischen Einflussparameters bei -10 lag. Für das Intervall $k \in [-0.2, 0.2]$ wird auch hier keine signifikante Auswirkung der heuristischen Funktion auf den Lernprozess gemessen.
In den theoretischen Überlegungen wurde der Abklingfaktor λ nicht berücksichtigt. In den Ergebnissen in den Tabellen 4.3, 4.4, 4.5 kann diese Abhängigkeit des heuristischen Einflussparameters k von den beiden Parametern γ und λ beobachtet werden. Diese Feststellung könnte in weiterführenden Arbeiten vertieft werden.
Die im Gitterweltproblem erzielte Tendenz einer positiven Auswirkung der heuristischen Funktion auf den Lernprozess hat eine Bestätigung im MCP gefunden. Das MCP ist viel komplexer, der Agent muss eine Kette aus Entscheidungen erlernen, die mehr als 100 Schritte beinhaltet. Die Einbindung der gut parametrisierten heuristischen Funktion in die Lernregel liefert eine stabile Beschleunigung des Lernprozesses für das MCP von 6% bis 10% im Vergleich zu den mit dem Standard-SARSA-Algorithmus erzielten Ergebnissen. Die Einbindung der heuristischen Funktion in der Aktionswahl bewirkte für das MCP nur wenige gute Ergebnisse. Eine Beschleunigung des Lernprozesses von bis zu 5% wurde für eine gut parametrisierte heuristische Funktion, angewendet für die Bildung der ϵ^*-gierigen Strategie, erzielt.

4.7 Ergebnisse für das Navigationsproblem

An dieser Stelle wird die Auswirkung der heuristischen Erweiterung auf die Lerngeschwindigkeit im Navigationsproblem für unterschiedliche Szenarien untersucht. Das Navigationsproblem wurde schon in Kapitel 2.4 eingeführt und der Roboter-Agent zur Lösung dieses Problems in Kapitel 2.5 vorgestellt.

Der Zustand s des Roboters ist ein dreidimensionaler Vektor (x, y, o), der aus der Position $(x, y) \in [A, B] \times [C, D]$ des Roboters im Raum und der Orientierung $o \in \{0°, 360°\}$ des Roboters besteht.
Die in diesem Kapitel aufgeführten Tabellen sind wie folgt aufgebaut. Die erste Spalte einer Tabelle beinhaltet den zu untersuchenden heuristischen Einflussparameter k. Der Parameter k liegt im Intervall von $[-1.0, -0.01]$ bei den Untersuchungen des SARSA*-Algorithmus bzw. $[-2, 1]$ bei der Untersuchung der ϵ^*-gierigen Strategie. Unterschiedliche Kombinationen des Parameters k werden in drei Umgebungen untersucht. In der zweiten Spalte der Tabelle ist der Lernfortschritt LP abgebildet. Dieser Wert wird relativ zum Standard-SARSA-Algorithmus ohne die heuristische Erweiterung gebildet. Die dritte Spalte gibt eine Übersicht über die Aussagekraft der Ergebnisse in der zweiten Spalte an. In der vierten und fünften Spalte ist das Ergebnis für das zweite Messkriterium Stoßzahl zusammengefasst. Der Wert in der vierten Spalte ist so definiert, dass je größer der relative Wert ist, desto geringer ist die Anzahl der aufgetretenen Kollisionen, und desto besser ist das Ergebnis der untersuchten Konfiguration. Auch dieser Wert wird in Relation zum Standard-SARSA-Algorithmus aufgestellt.

4.7.1 Definition der heuristischen Funktion

Die oben eingeführte Definition der heuristischen Funktion im Navigationsproblem basiert auf der Position des Roboters und stellt seine Distanz zum Ziel dar:

$$h(s) = \frac{\sqrt{(x - x^{Ziel})^2 + (y - y^{Ziel})^2}}{Norm}, \tag{4.35}$$

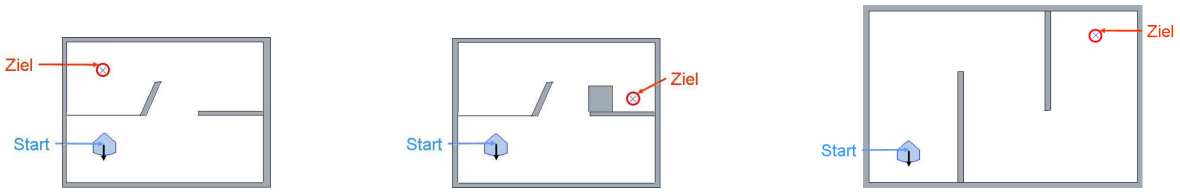
wobei $Norm \leftarrow \max_{(x,y)\in[A,B]\times[C,D]} \sqrt{(x - x^{Ziel})^2 + (y - y^{Ziel})^2}$. Die heuristische Funktion (4.35) wird im SARSA*-Algorithmus angewendet. Trotz des geringen Informationsgehalts und ohne die Möglichkeit in einem Schritt zwei Aktionen auseinanderzuhalten, wirkt sich diese heuristische Funktion auf die Lerngeschwindigkeit im SARSA*-Algorithmus positiv aus. Bei der Einbindung im erweiterten Lernschritt wirkt die heuristische Funktion auf die kompletten Entscheidungsketten, so dass der geringe Informationsgehalt kompensiert wird.
Die im MCP gemachte Beobachtung, dass die Anwendung der heuristischen Funktion für die Bildung der ϵ^*-gierigen Strategie sensibler auf die Definition und einen nicht ausreichenden Informationsgehalt der heuristischen Funktion reagiert, hat sich für das Navigationsproblem bestätigt. Für die ϵ^*-gierige Strategie und den gewählten diskreten Aktionsraum enthält die positionsbasierte heuristische Funktion nicht genügend Informationen, um die zwei von drei diskreten Aktionen "Drehe rechts" und "Drehe links" auseinander zu halten, da der Agent beim Ausführen dieser Aktionen an derselben Stelle bleibt. In der ϵ^*-gierigen Strategie führt die positionsbasierte heuristische Funktion zu keinen aussagekräftigen Ergebnissen. Für die ϵ^*-gierige Strategie wird eine alternative Definition der heuristischen Funktion (4.36) eingeführt.

$$h(s) = \sqrt{\left(\frac{(x - x^{Ziel})^2 + (y - y^{Ziel})^2}{Norm} + \frac{(o - o^{Ziel})}{Norm_o}\right) \cdot \sqrt{\frac{2}{3}}} \tag{4.36}$$

Für die ϵ^*-gierige Strategie wird das Signal des Sensors Kompass-Sinn in der heuristischen Funktion verarbeitet. Das Signal liefert die benötigte Differenz $(o - o^{Ziel})$ zwischen der aktuellen Orientierung o des Roboters und der benötigten Orientierung o^{Ziel} zum Ziel, siehe Gleichung (4.36). Durch $Norm_o = 2 \cdot 360°$ wird die Auswirkung des Orientierungsanteils in der heuristischen Funktion gesteuert. Der Parameter $\sqrt{\frac{2}{3}}$ setzt den Wertebereich der heuristischen Funktion auf das Intervall $[0, 1]$ zurück.

4.7.2 Verwendete Einstellungen und Szenarien

Die Parameter des SARSA-Algorithmus $\gamma = 0.9$, $\lambda = 0.8$, $\epsilon = 0.1$ und $\alpha = 0.2$ sind auf feste Werte gesetzt. Bei der Bestimmung des im vorliegenden Kapitel angewendeten Parametersatzes wurden die Empfehlungen für die beste Parametrisierung des SARSA-Algorithmus im Navigationsproblem aus der Diplomarbeit [Wolter, 2007] berücksichtigt, so dass der Standard-SARSA-Algorithmus unter günstigen Bedingungen lernte.
Die drei in diesem Kapitel verwendeten Szenarien für das Navigationsproblem sind in Abbildung 4.18 abgebildet. Alle drei Szenarien sind relativ einfach in ihrem Aufbau.
Mithilfe der Umgebung 1 in Abbildung 4.18(a) wird eine Einfahrt in einen anderen Raum simuliert. Das



(a) Umgebung 1 (b) Umgebung 2 (c) Umgebung 3

Abbildung 4.18: Einfache Beispielszenarien für das Navigationsproblem. Die Startposition und Startrichtung sind durch den Agenten und das Ziel durch den roten Kreis gekennzeichnet.

Ziel ist in diesem Szenario relativ einfach zu erreichen. Die Hauptaufgabe des Agenten liegt in der Navigation durch enge Pässe.
In Umgebung 2 in Abbildung 4.18(b) ist die Einfahrt in den Raum noch enger. Der Agent muss zusätzlich zu einer groben Navigation fein navigieren können, um das Ziel zu erreichen. Das Ziel wird in eine enge Einbuchtung platziert. Die Aufgabe des Agenten ist im Vergleich zur Umgebung 1 anspruchsvoller.
Die Umgebung 3 ist größer als die Umgebungen 1 und 2. Diese Umgebung simuliert eine Art Labyrinth bzw. Flur und ist hauptsächlich für eine Analyse des heuristischen Einflusses gedacht.
Die Umgebungen sind unterschiedlich aufgebaut. Die als normierte Abstandsfunktion bis zum Zielpunkt definierte heuristische Funktion kann in den einzelnen Zuständen für alle Umgebungen eine irreführende Information zur Zielnähe angeben. Die Information zur Zielnähe stimmt nicht immer mit der tatsächlich optimalen Strategie überein, da eine Wand zwischen dem Agenten und dem Ziel liegen kann.
Das im Navigationsproblem verwendete Tile-Coding beinhaltet die folgende Konfiguration für die verschiedenen Umgebungen: Umgebung 1 und 2: $10 \times 10 \times 24$ Teile, Umgebung 3: $14 \times 12 \times 24$ Teile, jeweils ein Tiling.

4.7.3 ϵ^*-gierige Strategie

Die Auswirkung des heuristischen Einflusses in der Bildung der Strategie auf die Lerngeschwindigkeit im Navigationsproblem wird an dieser Stelle untersucht. Die zur Bildung der ϵ^*-gierigen Strategie angewendete heuristische Funktion ist im dreidimensionalen Zustandsraum definiert, so dass die Orientierung des Agenten in die Berechnungsschritte mit einfließt, siehe Gleichung (4.36).
In Tabelle 4.6 sind die Ergebnisse der Anwendung der ϵ^*-gierigen Strategie in den Standard-SARSA-Algorithmus für das Navigationsproblem und die Umgebung 2 zusammengestellt. Der Lernschritt für die Adaption der Q-Funktion wird mit der Update-Regel (3.10) realisiert. Die Entscheidung über die nächste Aktion wird durch die ϵ^*-gierige Strategie getroffen, vergleiche Gleichung (4.1). In der Tabelle sind relative Werte im Vergleich zu den Ergebnissen ohne heuristischen Einfluss für unterschiedliche

Tabelle 4.6: Umgebung 2: Relative Veränderung der erweiterten Methoden, ϵ^*-gierige Strategie, $\gamma = 0.95, \lambda = 0.9, \alpha = 0.2, \epsilon = 0.1$, heuristische Funktion (4.36)

k	LP	t-Test LP	SZ	t-Test SZ
1	-12.78%	1	-9.81%	1
-0.5	23.71%	1	22.77%	1
-1	20.53%	1	18.72%	1
-2	23.74%	1	21.37%	1

Ausprägungen des heuristischen Einflussparameters $k \in \{1, -0.5, -1, -2\}$ zusammengefasst.
Der Parameter $k = 1.0$ liefert eine signifikante Verschlechterung in beiden Messkriterien, vergleiche erste Zeile in Tabelle 4.6. Die mithilfe positiver Parameter k aufgestellte Heuristik steht in Konflikt zur Problemstellung. Der am weitesten vom Zielzustand entfernte Zustand wird von der heuristischen Funktion als der beste Zustand bewertet.
Für einen korrekt eingestellten heuristischen Einflussparameter $k < 0$ wird eine Verbesserung des Lernfortschritts von bis zu 24% und der Stoßzahl von bis zu 21% erzielt, siehe Zeilen 2, 3 und 4 in der Tabelle. Alle drei Ausprägungen des heuristischen Einflussparameters $k \in \{-0.5, -1.0, -2.0\}$ weisen ähnlich signifikante Beschleunigungen des Lernprozesses und signifikante Verbesserungen des Messkriteriums Stoßzahl auf. Eine Korrelation wird zwischen der Verbesserung der Lerngeschwindigkeit und der Verbesserung der Stoßzahl beobachtet.
Für Umgebung 3 sind die erzielten Ergebnisse den Ergebnissen aus Umgebung 2 ähnlich. Eine signi-

Tabelle 4.7: Umgebung 3: Relative Veränderung der erweiterten Methoden, ϵ^*-gierige Strategie, $\gamma = 0.95, \lambda = 0.9, \alpha = 0.2, \epsilon = 0.1$, heuristische Funktion (4.36)

k	LP	t-Test LP	SZ	t-Test SZ
1	-30.69%	1	-14.58%	1
-0.5	14.89%	1	12.78%	1
-1	15.78%	1	13.4%	1
-2	18.8%	1	15.48%	1

fikante Verlangsamung des Lernprozesses von bis zu -31% und eine Steigerung der Anzahl der Stöße von bis zu -15% liefert ein ungeeignet definierter Parameter für den heuristischen Einfluss von $k = 1.0$, siehe erste Zeile in Tabelle 4.7. Eine Verbesserung der Lerngeschwindigkeit von bis zu 18% wird erzielt, siehe letzte Zeile in Tabelle 4.7. Dabei werden ähnliche Werte für die Beschleunigung des Lernprozesses für alle drei untersuchten Werte des heuristischen Einflussparameters $k \in \{-0.5, -1, -2\}$ gemessen. Im zweiten Messkriterium Stoßzahl wird auch eine signifikante Verbesserung von bis zu 15% gemessen.
Die Untersuchungen der Einbindung der heuristischen Funktion für die Bildung der ϵ^*-gierigen Strategie wurden für zwei unterschiedliche Umgebungen durchgeführt. Die Ergebnisse sind bei kleinen Veränderungen des Parameters k stabil. Die Erweiterung des Standard-SARSA-Algorithmus um die ϵ^*-gierige Strategie in Verbindung mit der heuristischen Funktion (4.36) wird für die Lösung des Navigationsproblems empfohlen.

4.7.4 SARSA*-Algorithmus

In den Tabellen 4.8, 4.9, 4.10 sind die Ergebnisse für die Anwendung des SARSA*-Algorithmus im Navigationsproblem für drei Szenarien zusammengefasst. Im SARSA*-Algorithmus wird der Lernschritt

(4.2), der um die heuristische Funktion (4.35) erweitert ist, angewendet. Die Auswirkung der heuristischen Funktion im Lernprozess auf die Beschleunigung des Lernprozesses wird in Abhängigkeit von der Stärke des heuristischen Einflusses untersucht. Der heuristische Einfluss wird durch den Parameter k reguliert.
Aus Tabelle 4.8 wird ersichtlich, dass die signifikanten Verbesserungen im Messkriterium Lernfortschritt

Tabelle 4.8: Umgebung 1: Relative Veränderung des erweiterten SARSA*-Algorithmus im Vergleich zum SARSA-Algorithmus, $\gamma = 0.9, \lambda = 0.8$

k	LP	t-Test LP	SZ	t-Test SZ
-0.01	4.28%	0	1.99%	0
-0.1	7%	1	2.45%	0
-0.2	9.89%	1	2.16%	0
-0.3	9.17%	1	-2.2%	0
-0.4	6.32%	1	-5.32%	0
-0.5	-1.07%	0	-13.12%	1
-0.6	-5.83%	0	-14.48%	1
-0.7	-7.72%	0	-22.15%	1
-0.8	-4.18%	0	-21.13%	1
-1	-6.17%	0	-25.83%	1

von bis zu knapp 10% für $\gamma = 0.9$ nur für die Parameter aus dem Intervall $k \in [-0.4, -0.1]$ gemessen werden. Für $k \in [-1; -0.6]$ wird eine Verschlechterung im Messkriterium Lernfortschritt gemessen, wobei diese Verschlechterung keine signifikante Aussagekraft hat. Mit dem Zuwachs der Auswirkung des heuristischen Anteils, also einem größeren Betrag des Parameters k, wird eine Verschlechterung des Messkriteriums Stoßzahl gemessen. Schon für einen Wert des heuristischen Einflussparameters von $k \leq -0.3$ wird eine Verschlechterung in diesem Messkriterium festgestellt. Für $k \leq -0.5$ ist dies sogar eine signifikante Verschlechterung, siehe 4. und 5. Spalte, Zeile 6 bis 10. Für den heuristischen Einflussparameter $k = -1.0$ wird zwar eine Verschlechterung des LP-Kriteriums von nur -6% gemessen. Aber eine Verschlechterung des LP-Kriteriums in Verbindung mit einer aussagekräftigen Verschlechterung der Stoßzahl weist darauf hin, dass für diese Umgebung eine zu starke Gewichtung der heuristischen Funktion störend ist.
Die Umgebung 2 ist im Vergleich zur Umgebung 1 komplexer. In Umgebung 2 ist die Verbesserung des Lernfortschritts (Beschleunigung des Lernprozesses) mit bis zu 14% sogar deutlicher als in Umgebung 1, siehe Tabelle 4.9. Für das Intervall $k \in [-0.9, -0.3]$ und $\gamma = 0.9$ wird eine signifikante Verbesserung des LP-Kriteriums gemessen. Hier wird für den Parameter k ein Wertebereich von $k \in [-1.0, -0.5]$ empfohlen. Das zweite Kriterium verhält sich ähnlich wie in Umgebung 1. Für das Kollisionskriterium wird eine signifikante Verschlechterung für die Parameter $k < -0.5$ gemessen.
Für Umgebung 3 sind die durchgeführten Versuche nicht so detailliert wie in den Umgebungen 1 und 2. Das Ziel der in Umgebung 3 durchgeführten Versuche war es, die in den Umgebungen 1 und 2 gemachten Beobachtungen und erkannte Tendenzen zu bestätigen. In allen durchgeführten Versuchen wird eine signifikante Beschleunigung der Lerngeschwindigkeit von bis zu 25% gemessen, siehe Tabelle 4.10, wobei wieder für $k \leq -0.8$ eine signifikante Verschlechterung der Stoßzahl gemessen wird.
Der empfohlene Wertebereich für den heuristischen Einflussparameter k bei Anwendung von $\gamma = 0.9$ und $\lambda = 0.8$ liegt für alle drei Umgebungen im Intervall $[-0.2, -0.5]$ und liefert stabil gute Ergebnisse. Für alle drei Umgebungen wird eine Beschleunigung des Lernprozesses bei Anwendung des SARSA*-Algorithmus von 10% bis 24% festgestellt. Die Einbindung der heuristischen Funktion in den Lernpro-

Tabelle 4.9: Umgebung 2: Relative Veränderung des erweiterten SARSA*-Algorithmus im Vergleich zum SARSA-Algorithmus, $\gamma = 0.9, \lambda = 0.8$

k	LP	t-Test LP	SZ	t-Test SZ
-0.01	3.36%	0	-0.34%	0
-0.1	4.44%	0	-2.12%	0
-0.3	11.01%	1	-1.66%	0
-0.4	13.62%	1	-2.18%	0
-0.5	8.96%	1	-8.21%	1
-0.6	11.87%	1	-7.99%	1
-0.7	14.19%	1	-6.39%	1
-0.8	12.72%	1	-10.36%	1
-0.9	11.9%	1	-11.98%	1
-1	7.62%	0	-17.05%	1
-1.1	3.28%	0	-20.78%	1

Tabelle 4.10: Umgebung 3: Relative Veränderung des erweiterten SARSA*-Algorithmus im Vergleich zum SARSA-Algorithmus, $\gamma = 0.9, \lambda = 0.8$

k	LP	t-Test LP	SZ	t-Test SZ
-0.3	22.28%	1	3.02%	1
-0.5	23.57%	1	-1.51%	0
-0.8	24.67%	1	-6.31%	1
-1	23.86%	1	-8.64%	1

zess, angewendet für das Navigationsproblem, wirkt sich negativ auf das zweite Messkriterium Stoßzahl aus.

4.8 Zusammenfassung

Im vorliegenden Kapitel wurden zwei Möglichkeiten die Lernmethode um die heuristische Funktion zu erweitern erläutert. Die erste Möglichkeit wird durch die Anwendung der heuristischen Funktion in der Bildung der angewendeten Strategie (ϵ^*-Strategie) realisiert. Die Update-Regel für die Adaption der Q-Funktion bleibt dabei gleich wie im Standard-SARSA-Algorithmus. Die zweite Möglichkeit wird durch die Einbindung der heuristischen Funktion in die Update-Regel realisiert und als SARSA*-Algorithmus bezeichnet.
Die vorgeschlagenen Erweiterungen werden zuerst auf Zulässigkeit analysiert. Rahmenbedingungen, in denen die vorgeschlagenen Erweiterungen nie schädlich sind, werden aufgestellt. Anschließend wurden auch die positiven Auswirkungen der vorgeschlagenen Erweiterungen in drei unterschiedlichen Aufgaben gemessen.
Es wurde eine Sensibilität der ϵ^*-gierigen Strategie gegen die Definition der heuristischen Funktion und ihren Informationsgehalt beobachtet. Für eine gut definierte heuristische Funktion wurden signifikante Verbesserungen der Lerngeschwindigkeit in allen untersuchten Problemstellungen gemessen. Eine Beschleunigung des Lernprozesses bei Anwendung der ϵ^*-gierigen Strategie im Gitterweltproblem von bis zu 12.5% und im MCP von bis zu 5% für die beste Parametrisierung wurde gemessen. Eine Beschleunigung des Lernprozesses von 24% im Navigationsproblem für Umgebung 2 bzw. 18% für Umgebung 3 weist auf die Nützlichkeit der Anwendung der ϵ^*-gierigen Strategie hin.

Das große Potenzial des SARSA*-Algorithmus hat sich in den einfachen Szenarien der Gitterwelt entfaltet. Eine Beschleunigung des Lernprozesses in beiden untersuchten Szenarien von 40% und 50% wurde gemessen. Für das MCP mit der besten Parametrisierung wurde eine Beschleunigung des Lernprozesses von nur 10% gemessen. Im Navigationsproblem wurde eine starke Abhängigkeit der Auswirkung der heuristischen Funktion vom Aufbau des Szenarios gemessen, eine Beschleunigung der Lerngeschwindigkeit von 10%, 14% und fast 25% für die beste Parametrisierung und Anwendung des SARSA*-Algorithmus wurde gemessen. In Anhang C wird ein weiteres Problem, das 3D-Mountain-Agent-Problem, aufgestellt. Die in diesem Problem erzielte Verbesserung von 2.75% weist auch auf die Problematik der Erweiterung hin. Wenn die heuristische Funktion keinen ausreichenden Informationsgehalt bietet, kann der erweiterte Algorithmus eine kleinere Effizienzsteigerung liefern als erwartet.

Kapitel 5

Lernen im Navigationsproblem auf niedriger Abstraktionsebene

Für eine erfolgreiche Lösung des Navigationsproblems ist es wichtig, dass der rationale Agent sowohl reaktive als auch planende Fähigkeiten besitzt. Die Eigenschaft selbständig neue Aktionen und Manöver auszuprobieren und auf dieser Grundlage zu erlernen ist daher entscheidend, zumal der Wissensstand des Agenten zu Beginn nicht vollständig oder gar nicht vorhanden ist.
Die Planung und Erstellung der Kette aus angefahrenen Zuständen und auszuführenden Aktionen kann auf unterschiedlichen Abstraktionsebenen erfolgen. Die ausgeführten Aktionen können selbst eine ganze Kette von einzelnen Aktionen beinhalten oder ausschließlich aus einer elementaren unteilbaren Aktion bestehen. In diesem Kapitel wird die erste im Rahmen der vorliegenden Arbeit erfolgte Realisierung eines rationalen, lernbasierten Agenten, der Navigationsprobleme löst, vorgestellt. Diese Realisierung basiert auf einem diskreten Aktionsraum, dessen Aktionen auf dem niedrigsten Abstraktionsniveau anzusiedeln sind. Ein solcher Agent wird im Folgenden als Reinforcement-Learning-Agent (RL-Agent) bezeichnet. Das Design dieses Agenten ist geprägt durch das zu lösende Navigationsproblem, die Gegebenheiten des realen Scorpion-Roboters, den vorgegebenen Aktionsraum sowie die an den Agenten gestellten Anforderungen. Dazu gehören unter anderem die Fähigkeiten zu planen, reaktiv zu handeln und Neues zu erlernen.
Das Lernen mithilfe des Reinforcement-Learning-Ansatzes basiert grundsätzlich auf den vom Agenten gemachten Fehlern. Erst nach einem Zusammenstoß mit einer Wand erfährt dieser, dass er das in Zukunft vermeiden sollte. In dem hier betrachteten Navigationsproblem haben solche Fehlentscheidungen allerdings schwerwiegende Konsequenzen. Ein Zusammenstoß mit einer Wand kann den Roboter vollständig außer Gefecht setzen. Das hier betrachtete Navigationsproblem ist für eine Umgebung definiert, in der auch Menschen agieren können. Daher ist der Aspekt der Personensicherheit bei der Realisierung des Agenten von großer Bedeutung. Um solchen Gefahrensituationen vorzubeugen, wird ein Schutzmechanismus entwickelt. Dieser funktioniert als eine Art Reflex. Der Mechanismus verarbeitet die rohen Daten aus den Abstandssensoren des Agenten und schickt in einer Gefahrensituation direkt Befehle an das Antriebssystem des Agenten, ohne dass ein Planungsschritt zwischengeschaltet wird.
In Kapitel 5.1 wird die Grundidee dieses Abschnittes geschildert. Die Anordnung der Bestandteile der RL-Agenten im rationalen Agent wird hervorgehoben. Abschließend wird ein kurzer Überblick über die schon gemachten Schritte auf dem Gebiet gegeben.
Der Standard-SARSA-Algorithmus wird um den oben beschriebenen Schutzmechanismus in Form von reaktiven Fähigkeiten erweitert. Im RL-Agenten wird die reaktive Fähigkeit als ein neuronales Netz realisiert. Die Diffusion des SARSA-Algorithmus mit reaktiven Fähigkeiten wird in Kapitel 5.2 eingeführt.
Dem Aspekt der Effizienzsteigerung des Lernprozesses wird in der vorliegenden Arbeit große Aufmerksamkeit gewidmet. Ein weiterer Vorschlag den Lernprozess durch die Einbindung von existierenden Informationen effizienter zu gestalten wird in Kapitel 5.3 vorgestellt. Der Vorschlag wird mit einem Vor-

initialisierungsschritt realisiert.
Die Analyse des entwickelten RL-Agenten und Bestimmung der optimalen Parameter des Agenten wird in Kapitel 5.4 behandelt.
Dieser Teil wird mit einer kleinen Zusammenfassung abgeschlossen, siehe Kapitel 5.5.

5.1 Lernen auf niedriger Abstraktionsebene

5.1.1 Grundidee

Der entwickelte rational agierende Agent soll je nach Situation sowohl reaktiv, basierend auf lokalen Informationen, als auch vorausschauend (planend), basierend auf globalen Informationen, handeln können. Durch Anwendung des SARSA-Algorithmus wird die Fähigkeit zum Planen und Lernen realisiert. Der auf dem SARSA-Algorithmus basierende Agent beinhaltet keine Möglichkeit dynamische Gegenstände zu umfahren. Die Anforderung der Reaktivität ist im Standard-SARSA-Algorithmus nicht erfüllt. Ein rationaler Agent, der das Navigationsproblem löst, soll aber fähig sein auf plötzlich auftauchende Objekte reagieren zu können, weil dynamische Objekte im Rahmen des Navigationsproblems nicht ausgeschlossen werden können. Der auf dem SARSA-Algorithmus basierende Agent wird erst dann als ein rationaler, lernender Agent bezeichnet, wenn die reaktive Fähigkeit[29] in die Grundstruktur des SARSA-Algorithmus des Agenten integriert ist.
Der rationale, auf dem SARSA-Algorithmus basierende und Navigationsprobleme lösende Agent erfährt den aktuellen Zustand[30] in der vorliegenden Arbeit über die Odometriedaten. Auf der anderen Seite sind die lokalen Informationen von den Abstandssensoren vorhanden und stehen dem Roboter-Agenten zur Verfügung. Diese Informationen nehmen an dem, durch den grundlegenden Algorithmus ausgeprägten Lernprozess nicht teil. Die im vorliegenden Kapitel verfolgte Idee zur Erweiterung des Standard-SARSA-Algorithmus lag auf der Hand. Die reaktive Fähigkeit, die die lokalen Informationen direkt in einen ausführbaren Steuerbefehl verarbeitet, wird eingeschaltet, wenn der Roboter-Agent einen kritischen Bereich betritt. Kurz vor einer möglichen Kollision mit der Wand oder anderen Gegenständen übernimmt die reaktive Fähigkeit die Steuerung und ändert die Fahrtrichtung des Roboters. Diese reaktive Fähigkeit ist auch mit menschlichen Reflexen vergleichbar. Wenn es ums Überleben geht, wird der Verstand meistens nicht gefragt, sondern der Körper operiert mit Reflexen, die vorgegeben sind.
Der RL-Agent ist ein auf dem SARSA-Algorithmus basierender, rationaler, lernender Agent, der Navigationsprobleme löst und auf Grundlage von Aktionen der niedrigen Abstraktionsebene lernt. Der RL-Agent soll eine optimale Sequenz aus Zustand-Aktions-Paaren erlernen, die den Agenten auf einem optimalen Weg vom Start- zum Zielpunkt führt. Die Grundstruktur des RL-Agenten sowie seine Bestandteile, wie Zustandsraum und Aktionsraum, sind vom eingeführten Roboter-Agenten abgeleitet, siehe Kapitel 2.5. Der RL-Agent basiert auf dem entwickelten kNN+SARSA-Algorithmus.

5.1.2 RL-Agent als lernender, rationaler Agent

Der RL-Agent ist in Abbildung 5.1 so dargestellt, dass gleiche Bestandteile des in Kapitel 2.2 eingeführten lernenden Agenten und des RL-Agenten durch gleiche Farben gekennzeichnet sind, vergleiche Abbildung 2.2.
Mit der Farbe rosa ist das Leistungselement gekennzeichnet. Dieses Element beinhaltet eine Wahr-

[29] Die Möglichkeiten zur Realisierung der gewünschten reaktiven Fähigkeiten mit dem Ziel den Agenten zu schützen sind weiter unten in Kapitel 6 ausführlich beschrieben.
[30] globale Information über die Position und Orientierung des Agenten im Raum

nehmungskomponente, eine Wissensverarbeitungskomponente und eine Ausführungskomponente. Mit dem Leistungselement wird die Entscheidung über die nächste Aktion auf die Aktuatoren vorgegeben. Das Leistungselement übernimmt den Planungsschritt. Die *Wissensverarbeitungskomponente* des RL-Agenten wird als Auswahl der besten Aktion ausgehend vom aktuellen Wissensstand realisiert. Das vorhandene Wissen wird durch die Q-Funktion repräsentiert. Die Q-Funktion beinhaltet indirekt ein über mehrere Episoden aufgebautes Modell der Umgebung. Der RL-Agent verwaltet und baut das Umgebungsmodell im Normalfall ohne jegliche Vorgaben auf. Wenn aber gewisse Vorstellungen über den Aufbau der Umgebung vorhanden sind, kann die Q-Funktion durch vorhandenes Wissen vorinitialisiert werden. Die *Ausführungskomponente* propagiert die gewählten Aktionen weiter. Gleichzeitig beinhaltet sie auch einen direkten Zugriff auf die Sensoren, so dass die reaktive Fähigkeit in notwendigen Situationen direkt ausführbar ist.
Das Lernelement ist durch die Farbe blau gekennzeichnet und berechnet den TD-Fehler. Mithilfe des TD-Fehlers wird die Q-Funktion an die aktuellen Messungen angepasst, siehe Gleichung (3.10).
Der Problemgenerator ist durch die Farbe grau gekennzeichnet und für den Explorationsschritt, also das Erkunden der Umgebung, verantwortlich. Dieser Schritt wird mithilfe des ϵ-Anteils der ϵ-gierigen Strategie realisiert. Der Problemgenerator ersetzt die durch das Leistungselement mit der Wahrscheinlichkeit ϵ bestimmte Aktion durch eine zufällige Aktion.
Die Kritik-Komponente ist mit der Farbe hellblau gekennzeichnet. Die vorgegebenen Leistungsstandards werden in der vorliegenden Arbeit in Form eines Belohnungsmodells formuliert. Die Kritik-Komponente gibt für den erreichten Zustand s_{t+1} und die ausgeführte Aktion a_t einen elementaren Belohnungswert r_t aus. Auf dieser Grundlage wird die Q-Funktion aufgebaut. Das Belohnungsmodell ist in Abhängigkeit von den verfolgten Zielen des zu lösenden Problems definiert und im Leistungsstandard formuliert.
Dem Roboter-Agenten stehen zwei Arten der Sensordaten zur Verfügung: die Odometriedaten und die

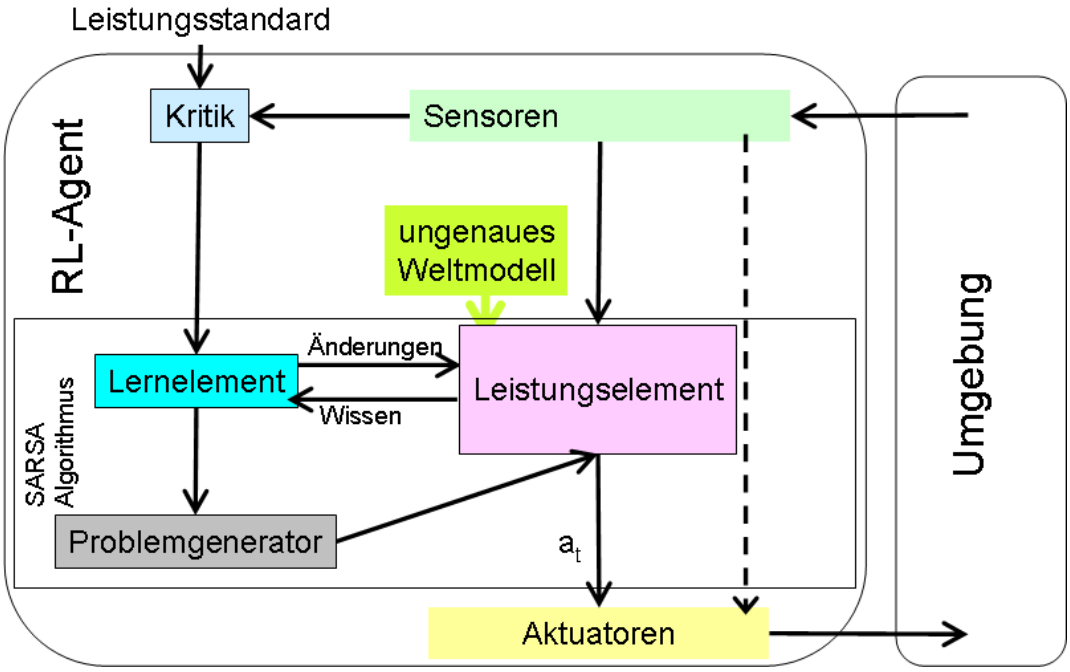


Abbildung 5.1: Darstellung des RL-Agenten im Sinne des lernenden Agenten, siehe Abbildung 2.2. Die grundlegenden Bestandteile des lernenden Agenten sind in beiden Abbildungen durch gleiche Farben gekennzeichnet.

Abstandsmessungen aus den IR-Sensoren. Auf Grundlage der Odometrie wird der im Leistungselement verwaltete, interne Zustand s_t bestimmt. Der lokale Abdruck der Umgebung, der durch IR-Sensoren erzielt wird, wird in den reaktiven Fähigkeiten verwendet. Die Aktuatoren des RL-Agenten verwalten die durch das Leistungselement bestimmten Aktionen des niedrigen Abstraktionsgrades. Der vom RL-Agent verwaltete Aktionsraum besteht aus drei diskreten Aktionen: (”Drehe rechts”, ”Drehe links”, ”Fahr geradeaus”), siehe auch Kapitel 2.5.3. Das von den Aktuatoren verwaltete neuronale Netz im RL-Agenten propagiert die Werte aus den IR-Sensoren und gibt zwei Werte, die Drehgeschwindigkeit und Vorwärtsgeschwindigkeit, für Befehle auf die Motorsteuerung des Roboters aus, vergleiche auch Kapitel 6.2.

5.1.3 Literaturübersicht

Der in der vorliegenden Arbeit eingeführte kNN+SARSA-Algorithmus behält im Gegensatz zur in [Noglik & Pauli, 2009] präsentierten Methode die Möglichkeit zum Lernen in allen Zustand-Aktions-Paaren bei.
Die Idee zur Anwendung der reaktiven und planenden Schritte und Verbindung dieser mit dem auf dem Reinforcement-Learning basierenden Lernschritt ist schon in der Arbeit ”Entwurf adaptiver lernender Roboter” in [Hamdi, 1999] beschrieben. Als Grundlage für das von Hamdi entwickelte Modell dienen eine Menge von konkurrierenden Prozessen und eine auf Prioritäten basierende Arbitrierungsstrategie. Diese Strategie wird dann mithilfe von RL-Methoden gelernt. Das entwickelte Modell wird für simulierte Umgebungen entwickelt. Der in der vorliegenden Arbeit entwickelte Roboter-Agent ist auf die Lösung eines Navigationsproblems ausgerichtet, beinhaltet eine Menge von Aktionen, die keine konkurrierenden Prozesse bilden, und wird auch in der Realität untersucht.
Die hier betrachtete Möglichkeit der Erweiterung des RL-Agenten besteht in der Einbindung eines existierenden, ungenauen Modells der Umgebung in den Lernprozess durch eine Initialisierung der Q-Funktion. Eine ähnliche Idee der Strukturextraktion aus bereits im Vorfeld vorhandenen Informationen und der Anwendung dieser Informationen in Form einer heuristischen Funktion ist in der Arbeit von Bianchi [Bianchi et al., 2008] beschrieben. In seiner Arbeit können die extrahierten, ungenauen Informationen allerdings nicht modifiziert werden, und der Lernprozess kann die Fehler der heuristischen Funktion nicht immer nachbessern. Die in der vorliegenden Arbeit vorgeschlagene Erweiterung besteht dagegen aus nur einem Vorinitialisierungsschritt für die Q-Funktion. Die weitere Adaption der Q-Funktion durch den Lernvorgang wird vorausgesetzt.

5.2 RL-Agent zur Roboternavigation

Der RL-Agent entsteht durch Erweiterung des SARSA-Algorithmus um eine reaktive Fähigkeit. Die reaktive Fähigkeit soll aber einen bestimmten Zweck erfüllen. Das verfolgte Ziel bei der Entwicklung der reaktiven Fähigkeit ist den Agenten so schnell wie es geht aus der Gefahrensituation zu steuern. Diese reaktive Fähigkeit wird durch das mit dem evolutionären Ansatz erzeugte neuronale Netz erreicht und wird als ”Hindernis umfahren” bezeichnet, siehe weiter unten Kapitel 6.2. Die durch das kNN erzielte reaktive Fähigkeit liegt auf dem anderen Abstraktionsniveau als die vom Standard-SARSA-Algorithmus verwendete Aktionsmenge. Die reaktive Fähigkeit kann nicht einfach als eine neue Aktion in der Aktionsmenge aufgenommen werden. Die Aktionen der Aktionsmenge sind konstante Werte, die an das Antriebssystem des Agenten übergeben werden und nicht vom lokalen Abdruck der Umgebung abhängen. Die beiden Aktionsarten sollen diffundieren.

5.2.1 Diffusion der Aktionen unterschiedlicher Abstraktionsebenen

Die Einbindung der reaktiven Fähigkeit in den Lernprozess wird durch die Einführung von zwei Schritten, dem Diffusions- und Projektionsschritt, realisiert. Die reaktive Fähigkeit ist für einen stetigen Aktionsraum definiert $a_{kNN} := (\omega, v) \in [-30°, 30°] \times [0, 20]$. Der RL-Agent lernt auf Grundlage der drei diskreten Aktionen.

$$a_t \in \{(15°, 0), (-15°, 0), (0°, 10)\}$$

Der Diffusionsschritt besteht aus der Übergabe der Steuerung in Gefahrensituationen an die reaktive Fähigkeit. Dieser Schritt bildet eine Art Diffusion der auf unterschiedlichen Abstraktionsebenen liegenden Aktionen und Informationen, die unterschiedliche Zweckbestimmungen haben (lokale und globale Informationsquellen).
Der Projektionsschritt, oder auch Diskretisierungsschritt genannt, projiziert die von der reaktiven Fähigkeit gewählten Befehle auf die dem RL-Agenten verwendeten diskreten Aktionen. Mithilfe des Projektionsschrittes wird vielen Problemen, die bei der Verwendung eines stetigen Aktionsraumes entstehen können, ausgewichen. Die Einbindung der reaktiven Fähigkeit kann unerwünschte Folgen haben, wenn Entscheidungen auf der Grundlage der Informationen getroffen werden, die nicht über den Lernprozess registriert werden. Zum Beispiel, wenn der Agent nahe an einer Wand und zur Wand hin ausgerichtet steht, und die Aktion "Fahr geradeaus" ausgeführt werden soll, dann wird der Agent noch näher an die Wand fahren. In diesem Fall übernimmt die reaktive Fähigkeit die Steuerung, um eine Kollision mit der Wand zu vermeiden. Wenn der Agent die reaktiv erzeugten Befehle nicht wahrnimmt und die Entscheidung "Fahr geradeaus" als etwas Gutes falsch interpretiert, dann wird die Q-Funktion anhand falscher Informationen aktualisiert, da der neue Zustand durch Einschalten eines kNNs, das nicht registriert wurde, erreicht wurde.
Drei Parameter steuern die Einbindung der reaktiven Fähigkeit in den Lernprozess. Auf diese Weise können unerwünschte Folgen vermieden werden. Der Parameter d_{Gefahr} besagt, ab wann die Situation als gefährlich einzustufen ist, so dass der Einsatz des Schutzmechanismus erwünscht ist. Der Parameter $Strafe$ erweitert das Belohnungsmodell. Dieser Parameter ist für die Bestrafung des Agenten verantwortlich, falls dieser sich in eine Gefahrensituation begeben hat. Der letzte Parameter $Skalierung$ steuert die Stärke des Steuerbefehls, den die reaktive Fähigkeit erzeugt. Mit diesem Parameter wird der durch das neuronale Netz erzeugte Steuerbefehl passend zum angewendeten Szenario skaliert.

5.2.2 kNN+SARSA-Algorithmus

Der kNN+SARSA-Algorithmus beinhaltet den oben angesprochenen Diffusions- und Projektionsschritt. Die Gefahrensituation wird durch den eingeführten Parameter d_{Gefahr} spezifiziert. Die Analyse der Gefahr erfolgt auf Grundlage der lokalen Abstandsmessungen, die die aktuelle lokale Situation relativ genau reflektieren. Wenn einer der Werte der IR-Sensoren den kritischen Wert unterschreitet, wird die Steuerung des Roboter-Agenten an die reaktive Fähigkeit[31] übergeben.
Der Algorithmus 5.1 enthält Erweiterungen in den Zeilen 17-21 im Vergleich zum SARSA-Algorithmus 3.2. In Zeile 17 wird die aktuelle Situation des Agenten geprüft. Die Situation des Agenten ist dann gefährlich, wenn der aktuell gemessene Wert d^t_{min} einen kritischen Wert d_{Gefahr} unterschreitet, siehe Zeile 17. Wenn die Situation als gefährlich erfasst wird, werden Steuerungsbefehle aus der kNN ausgelesen, siehe Zeile 18. Als Input erhält das künstliche neuronale Netz die lokalen Informationen von den Infrarotsensoren $d^t_1, \ldots, d^t_N$ zum aktuellen Zeitpunkt t. Als Output erzeugt das neuronale Netz

[31] Hier ist die reaktive Fähigkeit durch ein künstliches neuronales Netz realisiert.

Algorithmus 5.1 kNN+SARSA-Algorithmus mit erweiterter ϵ-gieriger Strategie

Initialisiere Parametervektor θ
$\mathbf{e} := \mathbf{0}$ ▷ Initialisiere Eligibility-Traces
for Für jede Episode $episode = 1, \ldots Episode$ **do**
    $t \leftarrow 0$
    $s_t \leftarrow Start$,
    Bestimme das Merkmalset $F(s_t)$ für den Zustand s_t
    for $a \in A(s_t)$ **do**
        $Q(s_t, a) \leftarrow \frac{1}{|F(s_t)|} \sum_{j \in F(s_t)} \theta_a(j)$
    $a_t \leftarrow \max_a Q(s_t, a)$
    for Für jeden Schritt t der Episode $episode$ **do**
        Führe die Aktion a_t aus, beobachte den Zustand s_{t+1} und die Belohnung r_t
        Bestimme das Merkmalset $F(s_{t+1})$ für den Zustand s_{t+1}
        for $a \in A(s_{t+1})$ **do**
            $Q(s_{t+1}, a) \leftarrow \frac{1}{|F|} \sum_{j \in F(s_{t+1})} \theta_a(j)$
        $d_1^t, \ldots, d_N^t \leftarrow IR - Sensoren$ ▷ Lese die aktuellen Werte der IR-Sensoren aus
        $d_{min}^t := \min(d_1^t, \ldots, d_N^t)$
        if $d_{min}^t < d_{Gefahr}$ **then**
            $(\omega, v) \leftarrow kNN(d_1^t, \ldots, d_N^t)$ ▷ kNN gibt Befehl auf Motorsteuerung als Antwort auf die lokalen Abdrücke der Umgebung
            $a_{kNN} \leftarrow Skalierung \cdot (\omega, v)$
            $a_{t+1} \leftarrow Projektion(a_{kNN})$
            $r_t \leftarrow r_t + Strafe$
        else
            $Zufall \leftarrow [0, 1]$ ▷ Generiere eine Zufallszahl aus dem Intervall $[0, 1]$
            if $Zufall < \epsilon$ **then** ▷ Bestimme die neue Aktion a_{t+1}
                $a_{t+1} \leftarrow \max_a Q(s_{t+1}, a)$
            else
                $a_{t+1} \leftarrow$ Zufällige Aktion aus $A(s_{t+1})$
        $\delta := r_t + \gamma Q(s_t, a_t) - Q(s_{t+1}, a_{t+1})$
        $\theta \leftarrow \theta + \alpha \cdot \delta \cdot \mathbf{e}$
        $\mathbf{e} := \lambda \mathbf{e}$
        for $j \in F(s_{t+1})$ **do**
            $e_{a_{t+1}}(j) = e_{a_{t+1}}(j) + 1.0$
        $t \leftarrow t + 1$

einen Steuerbefehl, der den Agenten vom Objekt wegsteuert. Die Stärke des mit einem kNN bestimmten Befehls wird hier mit dem Parameter $Skalierung$ gesteuert, und die Aktion a_{kNN} wird durch eine zusätzliche Skalierung aus dem erzielten Befehl gebildet, siehe Zeile 19.
Im Projektionsschritt wird für die durch ein kNN erzielte Aktion a_{kNN} die am nächsten liegende Aktion aus der vorhandenen diskreten Aktionsmenge bestimmt, siehe Zeile 20. Das Belohnungssignal wird in Gefahrensituationen auf den vordefinierten Wert $(r_t + Strafe)$ gesetzt, siehe Zeile 21. Durch den eingeführten Projektionsschritt kann der Lernprozess, siehe Zeilen 28-29, ohne Unterbrechungen fortgesetzt werden. Die Q-Funktion wird für die projizierte Aktion $a_{t+1} \leftarrow Projektion(a_{kNN})$ weiter adaptiert. Die vom RL-Agenten ausgeführte Aktionssequenz wird trotz des Eingriffs der fremden reaktiven Fähigkeit nicht abgebrochen. Das Eligibility-Traces-Konzept kann ohne Einschränkungen angewendet werden.
Eine genaue Beschreibung des Projektionsschrittes ist im Algorithmus 5.2 enthalten. Als Input erhält der

Algorithmus 5.2 Projektionsschritt für den kNN+SARSA-Algorithmus

```
1: procedure PROJEKTION(a_kNN)
2:     a_kNN = (ω, v)        ▷ beinhaltet zwei Einträge. Der erste Wert ist die Drehgeschwindigkeit, der zweite die Vorwärtsgeschwindigkeit.
3:     if |ω| < Projektion_const then
4:         return Fahr geradeaus
5:     else
6:         if ω > 0 then
7:             return Drehe links
8:         else
9:             return Drehe rechts
```

Projektionsalgorithmus die stetige Aktion a_{kNN}. Als Output berechnet der Algorithmus die am nächsten zur Aktion a_{kNN} liegende diskrete Aktion ”Fahr geradeaus”, ”Drehe links” oder ”Drehe rechts”. In Abhängigkeit vom konstanten Wert $Projektion_{const}$ wird entschieden, ob die ausgeführte Aktion noch als ”Fahr geradeaus” weiter berechnet wird, siehe Zeile 3. Im anderen Fall wird entschieden, ob die Aktion als ”Drehe links” oder ”Drehe rechts” weiter berechnet wird, siehe Zeilen 7- 9 in Algorithmus 5.2.

5.3 Vorinitialisierungsschritt des RL-Agenten

Wenn eine Vorstellung über die Umgebung existiert, kann diese Information in den Lernprozess einbezogen werden. Diese Art Umgebungsmodell muss nicht genau sein. Ungenaue Modelle der Umgebung können aus unterschiedlichen Quellen stammen. Für ein Navigationsproblem in geschlossenen Räumen existieren meistens Baupläne. Diese enthalten allerdings oft keine Informationen über die Position von z. B. Möbeln in den Räumen. Dynamische Komponenten der Umgebung wie Türen sind in solchen Bauplänen auch oft nicht vorhanden. Für Freiland-Anwendungen kann eventuell die Existenz von Karten angenommen werden. Die Baupläne oder die Karten werden dann als ungenaues Modell der Umgebung bezeichnet. Durch eine Initialisierung der Q-Funktion mit existierenden ungenauen Modellen der Umgebung kann der Lernprozess des RL-Agenten beschleunigt werden. Dieser Schritt wird durch den Kasten ”ungenaues Weltmodell” in Abbildung 5.1 symbolisiert.

5.3.1 Grundidee

Wenn die Voraussetzung der Existenz eines ungenauen Umgebungsmodells erfüllt ist, kann der in der vorliegenden Arbeit vorgestellte Vorinitialisierungsschritt durchgeführt werden. Die gesamte Erweiterung des SARSA-Algorithmus um ein ungenaues Umgebungsmodell läuft wie folgt ab. Zuerst wird aus dem ungenauen Modell ein Kraftfeld[32] mithilfe von Methoden der Bildverarbeitung erzeugt. Der Kraftfelder-Ansatz wurde ursprünglich zur Online-Kollisionsvermeidung entwickelt und kann direkt zur Navigation des Roboters verwendet werden. Die Q-Funktion wird mithilfe des Kraftfelds und des vorgeschlagenen Algorithmus zur Projektion des Kraftfelds vorinitialisiert. Diese vorinitialisierte Q-Funktion wird anschließend weiter mit dem SARSA-Algorithmus adaptiert.
Als Motivation zur Adaption des erstellten Kraftfelds werden folgende Gründe genannt. Der erste Grund liegt in der Ungenauigkeit der existierenden Baupläne. Die mit einem ungenauen Modell der Umgebung erzeugten Kraftfelder beinhalten auch Ungenauigkeiten. Ein zweiter Grund liegt in der durch die Kraftfelder erzeugten Lösung selbst. Die mit dem Kraftfelder-Ansatz erzeugte Lösung zeigt oft einen gravierenden Nachteil, der Agent gerät in lokale Extrempunkte, also Sackgassen, [Latombe, 1991, S. 295-355]. Das bedeutet, dass der Agent nicht aus jedem Punkt des Raumes mithilfe des erzeugten Kraftfelds zum Ziel navigieren kann. Stattdessen existieren Punkte im Raum von denen aus der Agent durch die erzeugten Kraftfelder zu den Sackgassen gesteuert wird. ”Eines der größten Probleme, welches im Zusammenhang mit Potentialfeldern auftritt, ist die Entstehung von lokalen Minima” [Deutsch, 2004]. Die Sackgassen entstehen an den Stellen, wo das Kraftfeld einen Nullvektor als Bewegungsvektor enthält, oder wo sich Zyklen bilden, die aus sich wiederholenden Aktionen bestehen. Wenn der Agent in den Sackgassen stecken bleibt, wird der vorgegebene Zielpunkt nicht erreicht und die gestellte Navigationsaufgabe nicht gelöst. Um das Problem zu umgehen, werden Strategien zum Verlassen eines solchen lokalen Extrempunkts notwendig. In der vorliegenden Arbeit wird anstatt der komplexen Strategien die Adaption der durch die Kraftfelder ermittelten Pläne angewendet. Die Probleme mit lokalen Extrempunkten werden anhand eines Beispiels in Kapitel 5.3.3 genauer geschildert.
Der Problematik der lokalen Minima im erzielten Kraftfeld wird mit Absicht wenig Aufmerksamkeit gewidmet. Die Bildung der Kraftfelder dient nur als Vorinitialisierungsinformation für den Agenten. Es wird keine analytische Untersuchung des Aufbaus der Räumlichkeiten vorgenommen. Jeder Punkt im Raum wird gleich behandelt. Für das Erzeugen der Kraftfelder wird der gleiche Algorithmus bei allen Testumgebungen angewendet.
Ein Nachteil der vorgeschlagenen Erweiterung liegt in der ineffizienten Nutzung der vorhandenen Informationen. Dieses hier verwendete, ungenaue Umgebungsmodell in Form von Bauplänen könnte auch viel effizienter genutzt werden, zum Beispiel durch Einbindung von solchen Planungsalgorithmen, wie dem A^*-Algorithmus. Für die Anwendung der Planungsalgorithmen ist das Vorhandensein eines Umgebungsplans eine notwendige Bedingung. Der Planungsalgorithmus ist nicht in dynamisch veränderten Umgebungen anwendbar. Ein Umgebungsplan ist nicht immer vorhanden. Aus diesen zwei Gründen ist die Anwendung der Planungsalgorithmen mit Beschränkungen und Schwierigkeiten verbunden.

5.3.2 Kraftfelder

Eine wichtige, oft getroffene Annahme bei dem Kraftfelder-Ansatz ist, dass der Roboter-Agent als Punktagent repräsentiert wird. Da aber ein späterer Lernvorgang zur Adaption der erstellten Kraftfelder vorgesehen ist, ist diese Annahme für die vorliegende Arbeit nicht von großer Bedeutung.

[32] Es wird der Kraftfelder-Ansatz aus der ursprünglich von Khatib entwickelten Potentialfeld-Methode [Khatib, 1986] verwendet.

Das ungenaue Umgebungsmodell wird als ein maßstabsgetreues Bild an einen Algorithmus übergeben, der aus dem Bild ein Kraftfeld erzeugt. Dabei wird der Zustandsraum bzw. der räumliche Anteil des Zustandsraumes diskretisiert und in einem digitalen Bild repräsentiert. Das ungenaue Umgebungsmodell beinhaltet in der hier beschriebenen Anwendung Informationen über die Position der Wände im Raum. Dieses Bild dient als Grundlage zur Bildung des negativen (abstoßenden) Kraftfelds. Wenn die Position des Zieles im Bild vorhanden ist, kann ein positives (anziehendes) Kraftfeld erstellt werden. Die Superposition der positiven und negativen Kraftfelder repräsentiert die gewünschte Kraftverteilung. Die Kraftverteilung beinhaltet für jeden Punkt eine Richtungsempfehlung. Diese Richtungsempfehlung leitet den Agenten im Idealfall in die Richtung des gewünschten Zieles, so dass der Agent im Idealfall, wenn er in jedem Punkt des Raumes der Richtungsempfehlung folgt, zum gewünschten Ziel navigiert wird. Die Superposition der erstellten positiven und negativen Kraftfelder bildet das für den Vorinitialisierungsschritt des RL-Agenten verwendete Kraftfeld.
Der Richtungsvektor wird für jeden Punkt (i, j) des Bildes als $(K_x(i, j), K_y(i, j))^T$ bezeichnet. Der vorhandene Umgebungsplan wird im Format eines Bildes mit einer vorgegebenen Größe von 400×400 repräsentiert. Mit der Farbe weiß sind starre Objekte, also Hindernisse, dargestellt. Die schwarzen Punkte sind Stellen, die frei von Objekten sind. Das Bild wird mit dem vorgegebenen Gauß-Operator gefaltet, um weiche Übergänge zwischen der Wand und dem freien Raum zu erzeugen. Das so erzeugte Bild $b(i, j) : \{1, 400\} \times \{1, 400\} \rightarrow [0, 1]$ beinhaltet das ungenaue Umgebungsmodell. Das Ziel im Bild wird als Punkt (z_x, z_y) repräsentiert.
Mithilfe eines Gradientenoperators[33] wird ausgehend vom vorhandenen Bild das abstoßende Kraftfeld erzeugt, siehe Gleichung (5.1). Das anziehende Kraftfeld wird durch einen normierten, in Richtung des Ziels zeigenden Vektor dargestellt, $(\frac{z_x}{||z||}, \frac{z_y}{||z||})$.

$$\begin{pmatrix} K_x(i,j) \\ K_y(i,j) \end{pmatrix} = - \begin{pmatrix} \frac{\partial b(i,j)}{\partial x} \\ \frac{\partial b(i,j)}{\partial y} \end{pmatrix} + \begin{pmatrix} \frac{z_x}{||z||} \\ \frac{z_y}{||z||} \end{pmatrix} \tag{5.1}$$

In Abbildung 5.2 ist das Ergebnis einer Superposition der anziehenden und abstoßenden Kraftfelder dar-

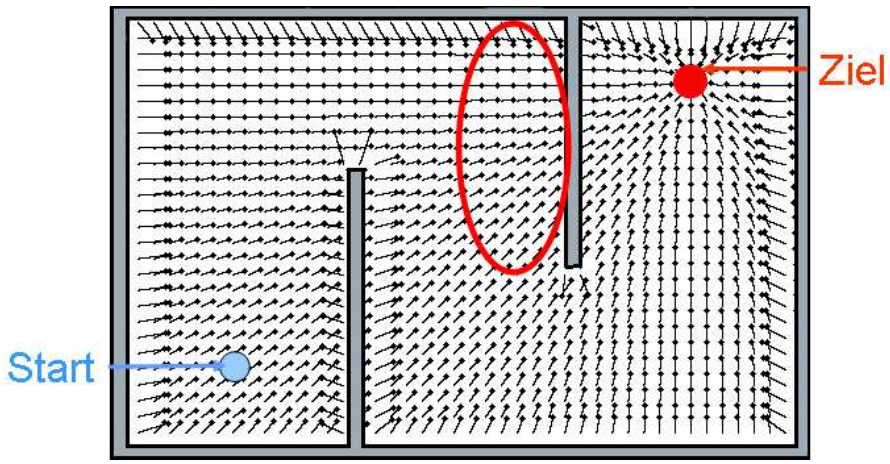


Abbildung 5.2: Beispiel für ein erzeugtes Kraftfeld. Mit der roten Ellipse ist der Bereich gekennzeichnet, wo ein lokales Minimum im Kraftfeld entstanden ist.

gestellt. Der rote Punkt markiert das Ziel. Die Vektoren, die mit dem Kraftfelder-Ansatz erzeugt worden sind, sind als Pfeile in Abbildung 5.2 dargestellt. Aus Gründen der Übersichtlichkeit sind die empfohle-

[33] Die Ableitung in der jeweiligen Richtung wird mit dem Sobel'schen Operator approximiert [Sobel & Feldman, 1968].

nen Richtungen nicht in jedem Punkt des Bildes dargestellt. Die Auswirkungen der abstoßenden Wände sind als Vektoren dargestellt.
Ein Beispiel für ein entstandenes lokales Minimum ist in Abbildung 5.2 mit einer roten Ellipse gekennzeichnet. Die lokalen Minima sind meistens in den Ecken von Nebenräumen zu finden, in denen der Agent stecken bleiben kann. Ein nicht adaptiertes Kraftfeld für das in der Abbildung dargestellte Szenario navigiert den Agenten vom vorgegebenen Startpunkt in eine durch eine rote Ellipse gekennzeichnete Ecke. Der Agent bleibt in der gekennzeichneten Ecke stecken, wenn er nur den durch das Kraftfeld vorgegebenen Plan ausführt.

5.3.3 Projektion der Kraftfelder auf die Q-Funktion

Die Q-Funktion wird mit dem erzeugten Kraftfeld initialisiert. Der Zustand des RL-Agenten besteht aus der Position des Agenten im Raum (x, y) und seiner Orientierung α^{ist}. Die Orientierung des Agenten α^{ist} kann im Intervall $[0, 360°]$ liegen. Das Kraftfeld ist ein Vektorfeld, das die empfohlenen Richtungen für jeden Punkt im Raum enthält. Für den Initialisierungsschritt der Q-Funktion erfolgt die Projektion des Kraftfelds auf die Q-Funktion mithilfe des Projektionsalgorithmus 5.3.
In Abbildung 5.3 sind die für den Projektionsalgorithmus wichtigen Werte wie die empfohlene Orientierung α^{soll}, die aus der empfohlenen Richtung des Kraftfelds für die Position (x, y) des Agenten bestimmt wird, und die aktuelle Orientierung des Agenten α^{ist}, die aus dem Orientierungsvektor des Agenten bestimmt wird, dargestellt. In Abbildung 5.3 ist die Orientierung beispielsweise auf $\alpha^{ist} = 180°$ gesetzt. Aus der durch das Kraftfeld vorgegebenen, empfohlenen Richtung wird die empfohlene Orientierung α^{soll} abgeleitet, wobei die Richtung ein vorgegebener, normierter Vektor ist. In der Abbildung hat die empfohlene Orientierung den Wert $\alpha^{soll} = 90°$.
Der Algorithmus 5.3 funktioniert folgendermaßen. Als Input dient ein Punkt (x, y) bzw. die Position des

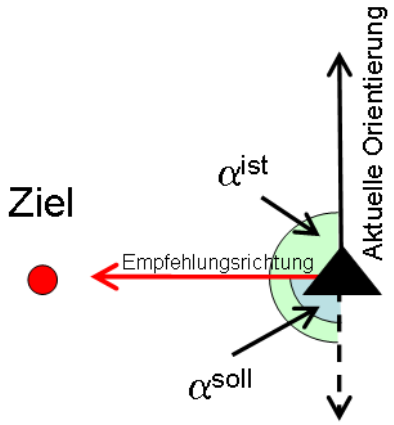


Abbildung 5.3: Aktuelle Orientierung α^{ist} und empfohlene Orientierung α^{soll} des Agenten

Agenten im Raum. Für diesen Punkt werden alle möglichen, diskretisierten Orientierungen des Agenten durchgegangen. Für diesen Punkt (x, y) existiert im Kraftfeld eine empfohlene Richtung, aus der der gewünschte Winkel α^{soll} abgeleitet wird, siehe Abbildung 5.3. Die Funktion $s(x, y, \alpha^{ist})$ liefert den Index im Merkmalvektor bzw. den Index für den aktuellen Zustand (x, y, α^{ist}) im Zustandsraum. Die Funktion $s(\cdot) : R^3 \rightarrow N$ wird mithilfe der ausgewählten Diskretisierungsmethode (Tile-Coding, Coarse-Coding oder RBF-Codierung) realisiert. Der Diskretisierungsschritt wird im vorliegenden Kapitel mit der Tile-Coding-Methode durchgeführt. Die Notwendigkeit der Diskretisierung und die Definition der Teile und

Algorithmus 5.3 Projektion des Kraftfelds auf die Q-Funktion

1: **procedure** PROJEKTION
2: Input Punkt (x, y) und α^{soll} - Winkel, der aus der empfohlenen Richtung des Kraftfelds für den Punkt (x, y) berechnet wird.
3: für alle diskretisierten Orientierungen α^{ist} ($\alpha^{ist} = i \cdot \delta$, $i = 0, 1, 2, \ldots, D$) ▷ $D := \frac{360^\circ}{\delta}$ die Anzahl der Teile im Orientierungsanteil
4: **if** $|\alpha^{ist} - \alpha^{soll}| < \delta$ **then**
5: $a^{soll} =$ *Fahr geradeaus*
6: **else**
7: **if** *Rechts*$(\alpha^{soll}, \alpha^{ist})$ **then**
8: $a^{soll} =$ *Drehe rechts*
9: **else**
10: $a^{soll} =$ *Drehe links*
11: Für alle Aktionen $a^{ist} \in A(s(x, y, \alpha^{ist}))$
12: **if** $a^{ist} == a^{soll}$ **then**
13: $Q(s(x, y, \alpha^{ist}), a^{ist}) = p$ bzw. $\theta_{a^{ist}}(s(x, y, \alpha^{ist})) = p$ ▷ $s(x, y, \alpha^{ist})$ Index für Zustand (x, y, α^{ist})
14: **else**
15: $Q(s(x, y, \alpha^{ist}), a^{ist}) = n$ bzw. $\theta_{a^{ist}}(s(x, y, \alpha^{ist})) = n$

Tilings sind in Kapitel 3 beschrieben. Mit $D := \frac{360^\circ}{\delta}$ wird die Feinheit der Diskretisierung, also die Anzahl der möglichen Teile, in der Komponente des Zustandes Orientierung $o \in [0, 360^\circ]$ ermittelt. Die Anzahl der Teile ist vorgegeben und der Diskretisierungsschritt δ wird aus dem vordefinierten Wert D berechnet.
In den Zeilen 4-10 wird die empfohlene *soll*-Aktion a^{soll} für den aktuellen Zustand bestimmt. In Zeile 4 wird durch Bildung der Differenz $|\alpha^{ist} - \alpha^{soll}|$ geprüft, ob für den Zustand (x, y, α^{ist}) die empfohlene Richtung aus dem Kraftfeld mit der tatsächlichen Richtung des Agenten übereinstimmt. Wenn dies der Fall ist, wird als empfohlene *soll*-Aktion in diesem Zustand "Fahr geradeaus" eingesetzt, siehe Zeile 5. Wenn dies nicht der Fall ist, wird für den Zustand (x, y, α^{ist}) die Aktion gewählt, die am schnellsten zur empfohlenen Richtung führt. Diese Prüfung findet in der Funktion $Rechts(\alpha^{soll}, \alpha^{ist})$ in Zeile 7 statt. Wenn die empfohlene Richtung durch Drehung nach rechts am schnellsten erreicht werden kann, wird als empfohlene *soll*-Aktion "Drehe rechts" angegeben a^{soll} =*Drehe rechts*, siehe Zeile 8, sonst a^{soll} =*Drehe links*.
Nach der Bestimmung der empfohlenen *soll*-Aktion für den Zustand (x, y, α^{ist}) wird die Q-Funktion mit dieser Information an der Stelle $s(x, y, \alpha^{ist})$ und allen möglichen Aktionen vorinitialisiert, siehe Zeilen 12-15. Wenn die *ist*-Aktion z. B. a^{ist} =*Drehe rechts* mit der vom Kraftfeld vorgegebenen a^{soll} =*Drehe rechts* übereinstimmt, wird ein positiver Eintrag p in der Q-Funktion an der Stelle $(s(x, y, \alpha^{ist}),$ *Drehe rechts*$)$ gemacht, d. h. $Q(s(x, y, \alpha^{ist}),$ *Drehe rechts*$) = p$, siehe Zeile 13. Für andere Aktionen wird ein negativer Eintrag n vorgenommen,

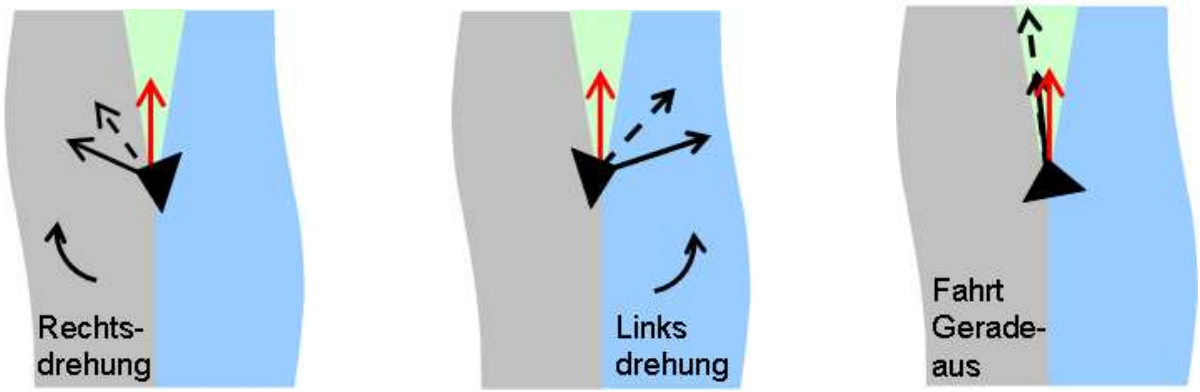
$$Q(s(x, y, o), \textit{Fahr geradeaus}) = n, Q(s(x, y, o), \textit{Drehe links}) = n,$$

siehe Zeile 15. Der Wert p steht für positiv und der Wert n steht für negativ. Für den Parametervektor erfolgt die Vorinitialisierung ähnlich:

$$\theta_{\textit{Drehe rechts}}(s(x, y, o)) = p,$$

$$\theta_{\{\textit{Fahr geradeaus, Drehe links}\}}(s(x,y,o)) = n.$$

Die Parameterkombination (p, n) wird für unterschiedliche Umgebungen in Kapitel 5.4.4 untersucht. Diese Parameterkombination soll in Relation zum verwendeten Belohnungsmodell ausgewählt werden.
In Abbildung 5.4 ist die Bestimmung der empfohlenen *soll*-Aktion a^{soll} aus dem Kraftfeld bildlich dargestellt. Der schwarze Pfeil stellt die aktuelle Orientierung des Agenten dar. Für einen Punkt sowie für die vom Kraftfeld entnommene, empfohlene, als roter Pfeil dargestellte Richtung ist die Projektion des Kraftfelds auf Einträge in der Q-Funktion durch Farben gekennzeichnet. Hier sind drei unterschiedliche Situationen der Orientierung des Agenten in Bezug auf die aus dem Kraftfeld entnommene, empfohlene Richtung dargestellt.
Wenn die Orientierung des Agenten im grauen Bereich liegt, siehe Abbildung 5.4(a), wird die Aktion ”Drehe rechts” empfohlen. Wenn die Orientierung des Agenten im blauen Bereich liegt, wird als *soll*-Aktion ”Drehe links” vorgegeben, siehe Abbildung 5.4(b). Wenn die Orientierung des Agenten im grünen Bereich liegt, dann wird als *soll*-Aktion ”Fahr geradeaus” vorgegeben, siehe Abbildung 5.4(c).
Die Funktion $Rechts(\alpha^{soll}, \alpha^{ist})$ wird folgendermaßen realisiert. Der Raum wird in Bezug auf die empfohlene Richtung $(a, b)^T$ in zwei Unterräume mithilfe der Orthogonalen zum Vektor der empfohlenen



(a) a^{soll} = Drehe rechts (b) a^{soll} = Drehe links (c) a^{soll} = Fahr geradeaus

Abbildung 5.4: Projektion des Kraftfelds auf die Q-Funktion. Der Farbton kennzeichnet die empfohlene *soll*-Aktion. Grau: ”Drehe rechts”, blau: ”Drehe links” und grün: ”Fahr geradeaus”. Der rote Pfeil ist die aus dem Kraftfeld entnommene, empfohlene Richtung, der schwarze Pfeil ist der aktuelle Orientierungsvektor des Agenten, der gestrichelte Pfeil ist der zukünftige Orientierungsvektor nach der Ausführung der empfohlenen *soll*-Aktion.

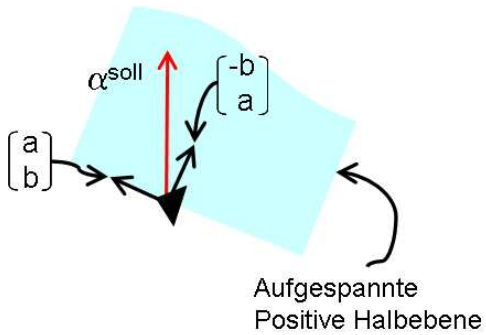
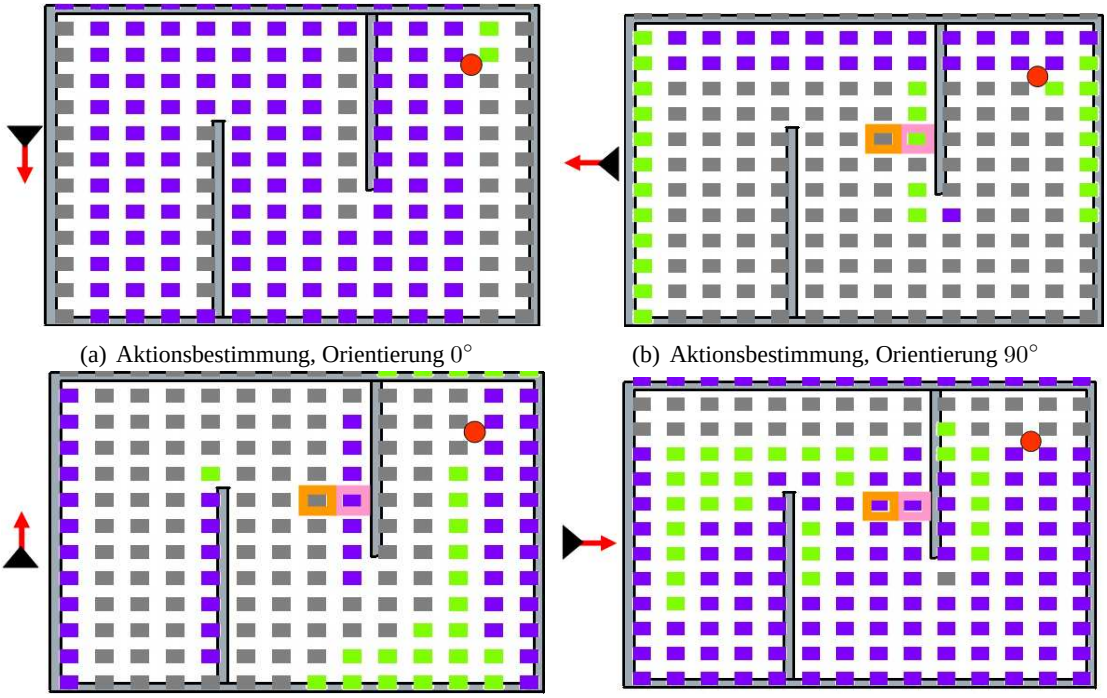


Abbildung 5.5: Prüfung der gewünschten Orientierung in Algorithmus 5.3

Richtung $(-b, a)^T$ aufgeteilt, siehe Abbildung 5.5. Wenn der durch das Kraftfeld erstellte Vektor der empfohlenen Richtung in der durch den Normalvektor aufgespannten positiven Halbebene liegt, siehe hellblaues Feld in Abbildung 5.5, gibt die Funktion $Rechts(\alpha^{soll}, \alpha^{ist})$ als empfohlene *soll*-Aktion "Drehe rechts", sonst "Drehe links" aus.
In Abbildung 5.6 ist bildlich die Q-Funktion dargestellt, die mithilfe des Projektionsalgorithmus vorinitialisiert worden ist. Der Zustandsraum des Agenten beinhaltet drei Dimensionen. Um die mit dem Projektionsschritt erzeugte Q-Funktion zu veranschaulichen, wird an dieser Stelle die Projektion des Zustandsraumes für vier unterschiedliche Orientierungen $(0°, 90°, 180°, 270°)$ dargestellt. In der Abbildung wird jeder Zustand durch ein Rechteck im Raum dargestellt. In Abbildung 5.6(a) ist der Zustandsraum mit einer Orientierung des Agenten von $0°$ abgebildet, in Abbildung 5.6(b) ist der Zustandsraum mit einer Orientierung des Agenten von $90°$ dargestellt. In Abbildung 5.6(c) ist der Zustandsraum mit einer Orientierung des Agenten von $180°$ und in Abbildung 5.6(d) mit einer Orientierung von $270°$ dargestellt. Die verschiedenen Orientierungen sind neben den einzelnen Abbildungen durch schwarze Pfeile gekennzeichnet.
Mithilfe der Farben werden die durch den Projektionsalgorithmus bestimmten *soll*-Aktionen für jeden Zustand des Raumes repräsentiert. Grauer Farbton bedeutet "Drehe rechts" als *soll*-Aktion, blauer Farbton bedeutet "Drehe links" als *soll*-Aktion, grüner Farbton bedeutet "Fahr geradeaus" als *soll*-Aktion.
In Abbildung 5.6 werden ein paar bereits geschilderte Probleme des Kraftfelder-Ansatzes deutlich. Das



(a) Aktionsbestimmung, Orientierung $0°$

(b) Aktionsbestimmung, Orientierung $90°$

(c) Aktionsbestimmung, Orientierung $180°$

(d) Aktionsbestimmung, Orientierung $270°$

Abbildung 5.6: Durch ein Kraftfeld vorinitialisierte Q-Funktion für vier diskrete Orientierungen. Der Pfeil neben jedem Teilbild zeigt die im Bild betrachtete Orientierung des Agenten an. Der Farbton signalisiert die $soll$-Aktion a^{soll}: grau = "Drehe rechts", blau = "Drehe links", grün = "Fahr geradeaus"

erste Problem entsteht durch den Projektionsschritt und die vorgenommene Diskretisierung des Raumes. Somit können falsche Empfehlungen für Zustände berechnet werden, die nahe an einem Hindernis liegen. Aufgrund der gewählten Diskretisierung entstehen fehlerhafte Empfehlungen des Kraftfelds und Fehler in der vorinitialisierten Q-Funktion. Für jeden Teil des diskretisierten Zustandsraums wird die

empfohlene *soll*-Aktion aus dem obersten linken Punkt des Kraftfelds entnommen. Das angesprochene Problem kann an einem Beispiel verdeutlicht werden. In Abbildung 5.6(b) ist für die Orientierung $90°$ an der linken Wand als empfohlene *soll*-Aktion "Fahr geradeaus" vorgegeben, siehe grüne Rechtecke. Die empfohlene *soll*-Aktion führt zur Kollision mit der Wand. Grund dafür ist der gewählte Diskretisierungsschritt. Der Agent geht davon aus, dass er sich außerhalb des Raumes befindet und will sich zunächst von der Wand entfernen.
Das zweite Problem ist das Problem der Entstehung von Sackgassen, in denen der Agent stecken bleibt. Dieses Problem wird anhand der ermittelten Kraftfelder, siehe Abbildung 5.6 genauer erläutert. Die Position des Zustandes, an dem das Beispiel beginnt, ist durch ein rosafarbenes Rechteck gekennzeichnet, eine Orientierung von $270°$ wird gewählt, siehe Abbildung 5.6(d). Bei dieser Orientierung ist der Agent zum Ziel gerichtet. Die Aktionen seien für das durchgeführte Beispiel wie folgt grob definiert: {Drehe rechts =($+90°$, 0), Drehe links = ($-90°$, 0), Fahr geradeaus = (0, 10) }. Auf der linken Seite der Abbildung ist die aktuelle Orientierung durch einen Pfeil gekennzeichnet. An der rosa umrandeten Stelle ist als empfohlene *soll*-Aktion "Drehe links" vorgegeben. Diese empfohlene *soll*-Aktion ist durch den blauen Farbton des Rechtecks zu erkennen. Diese empfohlene *soll*-Aktion führt den Agenten zur neuen Orientierung. Da die Aktionen nur grob definiert sind, ist die neue Orientierung des Agenten $180°$. Für diese Orientierung ist die empfohlene *soll*-Aktion wieder eine Drehung nach links. Auch hier ist die empfohlene *soll*-Aktion durch die blaue Farbe gekennzeichnet, siehe Abbildung 5.6(c). Nach der Ausführung der Aktion ist die neue Orientierung des Agenten $90°$. In diesem Zustand ist die empfohlene *soll*-Aktion "Fahr geradeaus". Diese Aktion ist durch die grüne Farbe gekennzeichnet, siehe Abbildung 5.6(b). Nach der Ausführung dieser Aktion geht der Agent in die neue Position über. Diese ist in Abbildung 5.6(b) durch eine orangene Umrandung gekennzeichnet. Im neuen Zustand ist als empfohlene *soll*-Aktion "Drehe rechts" vorgegeben. Diese Empfehlung ist in Abbildung 5.6(b) mit einem grauen Rechteck im orange umrandeten Zustand zu erkennen. Die neue Orientierung des Agenten ist dann $180°$. Für den orange umrandeten Zustand ist als empfohlene *soll*-Aktion "Drehe rechts" vorgegeben, siehe Abbildung 5.6(c). Die Orientierung des Agenten ändert sich auf $270°$. Für den aktuellen Zustand in Abbildung 5.6(d) wird als empfohlene *soll*-Aktion "Drehe rechts" vorgegeben. Dies ist am grauen Rechteck auf dem orange umrandeten Zustand zu erkennen. Damit entsteht ein Zyklus, aus dem der Agent nicht ohne einen weiteren Lernschritt herausfinden kann.

5.4 Ergebnisse für das Navigationsproblem mit dem RL-Agenten

Die Lernfähigkeit des RL-Agenten und die optimalen Bedingungen, unter denen der Lernvorgang am schnellsten vorangeht, werden in diesem Kapitel analysiert. Die Analyse des entwickelten RL-Agenten besteht aus der Bestimmung der optimalen Parameter des RL-Agenten und des Vorinitialisierungsschrittes. Die Lernqualität des Lernvorgangs, wird durch zwei Kriterien erfasst. Das erste Kriterium drückt die Lerngeschwindigkeit aus, das zweite Kriterium drückt die Sicherheit des Agenten in der Umgebung aus. Beide Kriterien wurden im Kapitel 3.6 eingeführt. Um die Aussagekraft der erzielten Ergebnisse zu verstärken, werden Versuche in mehreren unterschiedlich aufgebauten Szenarien durchgeführt.
Die reaktive Fähigkeit des RL-Agenten hat eine ähnliche Auswirkung wie das erstellte abstoßende Kraftfeld. Wenn der RL-Agent zu nah an einer Wand steht, wird der Agent vom angewendeten kNN weg vom Hindernis gesteuert. Der Agent wird aber auch durch das abstoßende Kraftfeld weg von den Wänden gesteuert. Zur Trennung beider Einflüsse wird die Vorinitialisierung mit dem Standard-SARSA-Algorithmus, der keine reaktive Fähigkeiten beinhaltet, verglichen.
In Kapitel 5.4.1 werden die für einen durchgeführten Versuch angewendeten Einstellungen und ein zu-

sätzliches Szenario für das Navigationsproblem eingeführt. Dieses Szenario ist aus dem realen Bauplan des Universitätsgebäudes entstanden.
In Kapitel 5.4.2 wird der Lernvorgang des RL-Agenten hinsichtlich der bis zum vernünftigen Ergebnis benötigten absoluten Schrittanzahl und Episodenanzahl diskutiert. Gleichzeitig werden hier auch Vor- und Nachteile des RL-Agenten erörtert.
Im Kapitel 5.4.3 wird der optimale Wert für den neuen Parameter $Skalierung$ des RL-Agenten bestimmt, siehe Zeile 19 im Algorithmus 5.1.
Die Erweiterung des SARSA-Algorithmus um den bereits erläuterten Vorinitialisierungsschritt wird in Kapitel 5.4.4 untersucht. Der Vorinitialisierungsschritt wird durch zwei Parameter p und n gesteuert, siehe Zeilen13 und 15 in Algorithmus 5.3. Die zwei Parameter korrelieren mit dem verwendeten Belohnungsmodell. Der optimale Parametersatz (p, n) wird für die dargestellte Erweiterung in diesem Unterkapitel ermittelt.
Die Kombination des RL-Agenten mit dem vorgeschlagenen Vorinitialisierungsschritt, sowie die Kombination des allgemeinen Konzepts des heuristischen Einflusses aus Kapitel 4 mit dem RL-Agenten werden bezüglich ihrer Auswirkung auf die Lerngeschwindigkeit in Kapitel 5.4.5 untersucht.

5.4.1 Verwendete Einstellungen und Szenarien

Zur Durchführung der Analyse des RL-Agenten werden die drei aus Kapitel 4.7.2 in Abbildung 4.18 dargestellten Szenarien im Navigationsproblem aufgestellt. Das in Abbildung 5.7 abgebildete Szenario wird zur Verifizierung der erzielten Ergebnisse verwendet.
Die *Umgebung BC* in Abbildung 5.7 ist ein Ausschnitt aus dem Bauplan des Universitätsgebäudes und stellt ein Beispiel für eine komplexe Welt mit sehr vielen Zuständen (Zustandsraum wird in Teile zerlegt) dar. Die Umgebung BC hat $\underbrace{40}_{x-Achse} \times \underbrace{31}_{y-Achse} \times \underbrace{24}_{Orientierung}$ Teile und ein Tiling. Im Vergleich

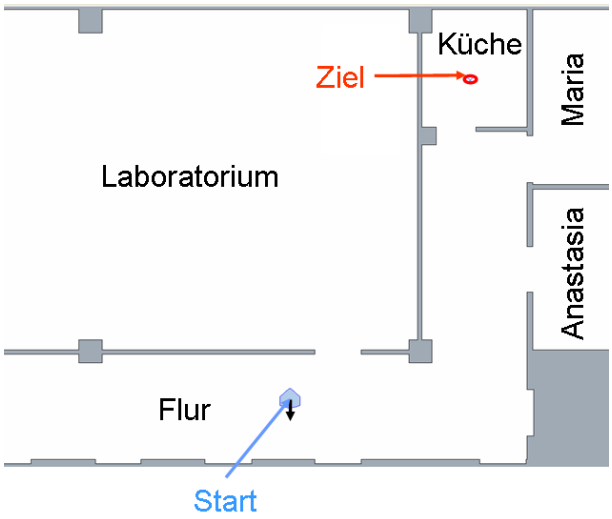


Abbildung 5.7: Umgebung BC. Der Bauplan eines Ausschnitts einer Etage des Universitätsgebäudes wird als ein komplexeres Szenario für das Navigationsproblem verwendet. Der Agent befindet sich in seiner Startposition und -ausrichtung. Der rote Kreis ist das Ziel.

dazu beinhaltet die Umgebung 3 in Abbildung 4.18 insgesamt $14 \times 12 \times 24$ Teile, vergleiche Kapitel 3.7.3. Das Feld der Umgebung BC beinhaltet damit etwa siebenmal so viele Zustände wie das Feld der Umgebung 3.

In der vorliegenden Arbeit sind viele unterschiedliche Ergebnisse erzielt worden, die unterschiedlichen Zwecken dienen. Als Vergleichsmaß dienen in allen Fällen Ergebnisse, die mit dem Standard-SARSA-Algorithmus erzielt worden sind. Auf diese Weise können Ergebnisse kapitelübergreifend miteinander verglichen werden.
Die für das Lernen verwendeten Parameter werden standardmäßig wie folgt festgelegt: $\alpha = 0.2, \epsilon = 0.1, \gamma = 0.9, \lambda = 0.8$, vergleiche dazu Kapitel 3.3.3. Die Anzahl der Episoden pro durchgeführten Versuch beträgt 250, die maximale Anzahl an Schritten pro Episode wird auf 10000 gesetzt. Jeder Versuch wird mindestens 30 Mal durchgeführt. Der RL-Agent ist aus dem Standard-SARSA-Algorithmus abgeleitet, und die für den SARSA-Algorithmus benötigten Parameter werden standardmäßig gesetzt. Als Codierungs- bzw. Diskretisierungsmethode wird Tile-Coding gewählt. Die Größe der Teile wird bei allen Szenarien beibehalten. Unterschiedlich große Szenarien haben damit auch eine unterschiedliche Anzahl an Teilen im Lernprozess. Überall wird nur ein Tiling verwendet.
Das Belohnungsmodell aller Versuche in diesem Abschnitt sieht wie folgt aus: Der Agent erhält für jeden Schritt eine Bestrafung von $r_t = -2$, für das Erreichen des Zieles eine Sonderbelohnung von $r_{Ziel} = 1000$ und für jede Kollision eine Sonderbestrafung von $r_{Kollision} = -20$.
An dieser Stelle wird der Aufbau der meisten im vorliegenden Abschnitt enthaltenen Tabellen kurz erläutert. Die untersuchten Parameter stehen jeweils in der ersten Spalte der Tabellen 5.3, 5.4, 5.5 und 5.6 in Kapitel 5.4.3 sowie in den Tabellen 5.7, 5.8, 5.9, 5.10 in Kapitel 5.4.4. Die zweite Spalte gibt die relative Veränderung des Lernfortschritts (LP), siehe Kapitel 3.6, im Vergleich zu den Ergebnissen des Standard-SARSA-Algorithmus, also ohne jegliche Erweiterungen, an. Die dritte Spalte beziffert die Signifikanz der Veränderung des Lernfortschritts. Die Aussagekraft (Signifikanz) wird auf Grundlage des t-Tests, siehe Kapitel 3.6, festgelegt. In der vierten Spalte wird die gemessene, relative Veränderung des Messkriteriums Stoßzahl im Vergleich zur Standard-Methode angegeben. Die fünfte Spalte gibt Auskunft über die Signifikanz der Veränderung der Stoßzahl.

5.4.2 Analyse des RL-Agenten

Der RL-Agent beinhaltet zusätzliche Parameter zum Einbinden der reaktiven Fähigkeiten. Der Wert $Projektion_{const}$ wird in Algorithmus 5.2 zur Diskretisierung der durch das kNN erzielten Aktion verwendet. Der Parameter d_{Gefahr} gibt den Wert vor, ab wann die Situation so kritisch ist, dass das künstliche neuronale Netz die Steuerung übernehmen soll. Der Parameter $Strafe$ gibt an, wie stark der Agent dafür bestraft werden soll, dass es zu solchen kritischen Situationen gekommen ist. Der Parameter $Skalierung$ justiert die mit neuronalen Netzen erzeugten Steuerungsbefehle für den RL-Agenten nach. Die Parameter des RL-Agenten wie der Diskretisierungsparameter $Projektion_{const}$, Erkennung der Gefahrensituation d_{Gefahr} oder Erweiterung des Belohnungsmodells $Strafe$ wurden für eine ähnliche Erweiterung des Standard-SARSA-Algorithmus (ohne Fortsetzung des Lernprozesses) systematisch von der Autorin in [Noglik & Pauli, 2009] untersucht. Die optimalen Parameter aus dieser Analyse sind $d_{Gefahr} = 25, Strafe = -2, Projektion_{const} = 8$. Gleiche Werte werden auch bei der hier durchgeführten Analyse verwendet. Der Parameter $Skalierung$ wird in diesem Abschnitt kalibriert.
Die durch ein neuronales Netz realisierte reaktive Fähigkeit soll den Agenten so schnell wie möglich aus der Gefahrensituation hinaussteuern. Die Szenarien für das Trainieren dieser Fähigkeit, sowie die verwendete Fitnessfunktion sind in Kapitel 6.2 beschrieben. Das für die Versuche eingebundene künstliche neuronale Netz ist in Abbildung 6.5 beispielhaft für die Aktion ”Hindernis umfahren” dargestellt. Die verwendeten Szenarien zur Erstellung des neuronalen Netzes unterscheiden sich stark von den Umgebungen, in denen der RL-Agent angewendet wird. Die Diskretisierung des Zustandsraumes trägt dazu bei, dass die schwachen Veränderungen des Zustandes des Agenten, nicht vom Lernprozess registriert

werden. Das Einbinden der reaktiven Fähigkeit benötigt eine Art Kalibrierung. Der vom Netz erzeugte Befehl wird mit dem Parameter $Skalierung$ skaliert und angewendet, siehe Zeile 19 in Algorithmus 5.1.
In den Tabellen 5.1 und 5.2 sind die Ergebnisse aus Umgebung 2 für unterschiedliche Werte des Para-

Tabelle 5.1: RL-Agent, Umgebung 2, absolute Werte für den Lernfortschritt und die Stoßzahl für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	Varianz LP	SZ	Netzwerkanwendungen
0	399.47	65.3554	4.04	0
1	395.62	81.6443	2.12	29.49
1.5	270.39	37.1724	1.64	18.11
2	281.47	32.41	1.94	17.02

Tabelle 5.2: RL-Agent, Umgebung 3, absolute Werte für den Lernfortschritt und die Stoßzahl für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	Varianz LP	SZ	Netzwerkanwendungen
0	1769.79	130.975	9.6948	0
1	1410.1	174.388	4.61	62.03
1.3	1275.37	139.034	4.06	48.03
1.5	1298.64	98.6849	4.37	45.09
2	1216.7	103.264 2	3.93	36.05

meters $Skalierung$ zusammengefasst. Der RL-Agent besitzt mehrere variable Parameter, deren Werte seinen Lernprozess beeinflussen. Im vorliegenden Unterkapitel wird der Parameter $Skalierung$ genauer untersucht. Der Parameter $Skalierung \geq 1.0$ verstärkt den durch das kNN erzeugten Steuerbefehl.
In den Tabellen 5.1 und 5.2 sind die beiden verwendeten Messkriterien, Lernfortschritt und Stoßzahl, als absolute Werte für die durchgeführten Versuche zusammengefasst, siehe 2. und 4. Spalte. Um eine Vorstellung über die Stabilität der durchgeführten Versuche zu bekommen, wird die Varianz der Werte des Lernfortschritts berechnet. Die fünfte Zeile gibt die durchschnittliche Anzahl der Aufrufe des neuronalen Netzes, also Netzwerkanwendungen, an. Diese Werte werden zur besseren Vorstellung über den Verlauf des Lernprozesses zusammengestellt.
Je stärker der durch das neuronale Netz erzeugte Befehl skaliert wird, desto weniger Aufrufe der reaktiven Fähigkeit werden benötigt. Wenn der Parameter $Skalierung$ den Wert 1 annimmt, wird für beide Umgebungen die maximale Anzahl an Netzwerkaufrufen erzielt. In diesem Fall wird für Umgebung 2 die reaktive Fähigkeit im Durchschnitt während des Lernprozesses in jeder Episode 29.49 Mal aufgerufen, für Umgebung 3 liegt dieser Wert bei 62.03, vergleiche Zeile 2 in beiden Tabellen. Wenn der Befehl stärker skaliert wird, werden deutlich weniger Netzwerkaufrufe benötigt. Für Umgebung 2 und $Skalierung = 2$ liegt dieses Maß bei einem Wert von 17.02, für Umgebung 3 und $Skalierung = 2$ wird die reaktive Fähigkeit während des Lernprozesses in jeder Episode im Durchschnitt 36.05 Mal aufgerufen. Der Agent wird deutlich schneller aus der Gefahrensituation gesteuert.
Die Anzahl der Zusammenstöße mit einer Wand wird weniger von einer stärkeren Skalierung des Steuerbefehls beeinflusst, vergleiche die Zeilen 1 bis 4 in Tabelle 5.1 bzw. Zeilen 1 bis 5 in Tabelle 5.2. Die Einbindung der reaktiven Fähigkeit erzielt auch eine Beschleunigung des Lernprozesses. Diese Beobachtung basiert auf dem Kriterium Lernfortschritt und wird im nächsten Kapitel genauer untersucht.
Für Umgebung 2 werden beim Versuch mit dem am stärksten skalierten Netz im Durchschnitt nur 1.9 Stöße pro Episode beobachtet. Zum Vergleich dazu liegt der Durchschnitt bei dem Versuch ohne Ein-

bindung der reaktiven Fähigkeit (in der Tabelle als $Skalierung = 0$ bezeichnet) bei 3.93 Stößen pro Episode.
In Umgebung 3 verringerte sich der Durchschnitt der Stöße mit Einbindung im Vergleich zu ohne Einbindung der reaktiven Fähigkeit um 5.76 Punkte, von 9.69 auf 3.93, vergleiche 1. und 5. Zeile in Tabelle 5.2.
Aus den erzielten Ergebnissen in beiden Tabellen wird ersichtlich, dass sich die gemittelte Anzahl der Kollisionen in etwa halbiert, wenn der Agent die reaktive Fähigkeit benutzt. An dieser Stelle kann beobachtet werden, dass das Verfahren trotz des eingebauten Schutzmechanismus Kollisionen mit Wänden vermindern aber nicht verhindern kann. Die Anzahl der Zusammenstöße kann durch eine Erhöhung des Wertes des Parameters d_{Gefahr} weiter reduziert werden. In diesem Fall besteht allerdings die Gefahr, dass der Lernprozess durch die Einmischung der reaktiven Fähigkeit zu stark beeinflusst wird. Eine folgenreiche Konsequenz dessen ist, dass der Agent sich ausschließlich um Kollisionsvermeidung kümmert. Der Lernprozess, bei dem die Kette aus Entscheidungen für das Erreichen des Ziels im Navigationsproblem erlernt werden soll, bleibt in diesem Fall auf der Strecke. Daher sollten gewisse Kosten in Anspruch genommen werden, um den Lernprozess fortsetzen zu können.
Für Umgebung 2 werden durchschnittlich 281 Lernschritte pro Episode benötigt. Für Umgebung 3, die im Vergleich zur Umgebung 1 und Umgebung 2 etwas größer ist, werden 1216 Lernschritte benötigt. Dies ist mehr als viermal so viel. Es ist zu erwarten, dass bei einem noch größeren Szenario wie der Umgebung BC die Anzahl der benötigten Lernschritte ins Unermessliche steigt.
Mit dem hier vorgeschlagenen RL-Agenten können nur Probleme in kleineren und einfacheren Szenarien gelöst werden. Für Aufgaben in einem größeren Bewegungsareal ist dieser schlecht geeignet. Daher ist der RL-Agent als eine Art Machbarkeitsstudie zu betrachten und sollte ausschließlich für kleinere Zustandsräume wie in Umgebung 1 und 2 angewendet werden.

5.4.3 Optimale Parametrisierung des RL-Agenten

Die ersten Tendenzen, wie sich der Parameter $Skalierung$ im RL-Agenten auswirkt sind im vorherigen Kapitel erkannt worden. In diesem Abschnitt werden schon die relativen Werte aufgestellt, die durch den Vergleich mit der Standard-SARSA-Methode berechnet werden. Das Ziel für diesen Abschnitt ist es einen optimalen Parameter $Skalierung$ für den RL-Agenten zu finden. Die Versuche werden in vier unterschiedlichen Umgebungen und für unterschiedliche Werte des Skalierungsparameters $Skalierung \in \{1, 1.3, 1.5, 2.0, 3.0\}$ durchgeführt, siehe Abbildung 4.18 und 5.7. In Tabelle 5.3 sind die Ergebnisse aus Umgebung 1 für unterschiedliche Werte des Parameters $Skalierung = \{1.0, 1.5, 2.0, 3.0\}$ dargestellt. Für einen Wert von $Skalierung = 3.0$ werden mit Abstand die schwächsten Ergebnisse für beide Messkriterien erzielt, siehe letzte Zeile der Tabelle 5.4. Dieses Ergebnis wird auch in der etwas komplexeren Umgebung 2 bestätigt. Für diesen Wert ($Skalierung = 3.0$) wird sogar eine, allerdings nicht signifikante, Verschlechterung beim Messkriterium LP im Vergleich zum Standard-SARSA-Algorithmus gemessen, siehe letzte Zeile der Tabelle 5.4. Auf Grundlage der gemachten Beobachtungen wird die Aussage getroffen, dass der optimale Wert für den Parameter $Skalierung$ für das verwendete kNN im Intervall $[1 : 2]$ liegt.
Für alle untersuchten Werte des Parameters $Skalierung$ liegen in Umgebung 1 signifikante Verbesserungen hinsichtlich beider Messkriterien vor, vergleiche Tabelle 5.3. In dieser Umgebung wird eine maximale Verbesserung des Lernfortschritts von bis zu 31.98% erzielt, siehe 3. Zeile für den Wert $Skalierung = 2.0$. Für den gleichen Wert $Skalierung = 2.0$ wird auch die maximale Verbesserung der Stoßzahl von 57.82% erzielt.
In Umgebung 1 für den Wert $Skalierung = 1.0$ bzw. für den Wert $Skalierung = 1.5$ wird eine Ver-

besserung des LP-Kriteriums von 22.54% bzw. von 25.54% gemessen. Diese Werte sind um knapp 10 Prozentpunkte bzw. 6.5 Prozentpunkte schlechter als das beste erzielte Ergebnis für $Skalierung = 2.0$, siehe Tabelle 5.3.
Der Unterschied im Messkriterium Stoßzahl ist nicht so gravierend, 54.72% für $Skalierung = 1.0$ im Vergleich zu 57.82% für $Skalierung = 2.0$, also ein Unterschied von etwa 3 Prozentpunkten. Für den Wert $Skalierung = 1.5$ liegt die Verbesserung der Stoßzahl bei 57.17%, also ein Unterschied von weniger als 1 Prozentpunkt zum besten Ergebnis, das mit $Skalierung = 2.0$ erzielt wird.
Die für Umgebung 1 gemachte Beobachtung einer nichtlinearen Abhängigkeit der erzielten Verbesserung des Lernfortschritts und der Stoßzahl wird auch in allen verwendeten Umgebungen 2, 3 und BC gemacht.
Für Umgebung 2 ist der Unterschied zwischen den gemessenen Werten des Lernfortschritts für unter-

Tabelle 5.3: Umgebung 1, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten im Vergleich zum Standard-SARSA-Algorithmus für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	t-Test LP	SZ	t-Test SZ
1	22.54%	1	54.72%	1
1.5	25.54%	1	57.17%	1
2	31.98%	1	57.82%	1
3	11.17%	1	19.83%	1

schiedliche Werte des Parameters $Skalierung \in [1; 2]$ noch größer. Für den Wert $Skalierung = 1.0$ wird keine signifikante Verbesserung des Lernfortschritts im Vergleich zum Standard-SARSA-Algorithmus gemessen, dagegen wird eine deutliche Verbesserung der Stoßzahl von knapp 50% gemessen, siehe 1. Zeile der Tabelle 5.4.
Für die Werte $Skalierung = 1.3$ und $Skalierung = 1.5$ wird eine signifikante Verbesserung beider Messkriterien beobachtet, für den Lernfortschritt sind es 31% für $Skalierung = 1.3$ bzw. 37% für $Skalierung = 1.5$, für die Stoßzahl wiederum 61% bzw. 62%. Unerwartete Werte liefert der Wert $Skalierung = 2.0$. Für diesen Parameterwert wird eine ähnliche Verbesserung des LP-Kriteriums (31.48%) erzielt wie für den Wert $Skalierung = 1.3$. Hingegen liegt die Verbesserung der Stoßzahl bei 52.83%, was um etwa 8 Prozentpunkte schlechter ist als für den Wert $Skalierung = 1.3$. Das Ziel in Umgebung 2 liegt in einem engen Korridor. Der Schutzmechanismus übernimmt die Steuerung, wenn der Agent in den engen Korridor fährt, so dass eine richtige Kalibrierung sehr wichtig für diese Umgebung ist.
Für Umgebung 3 werden fast die gleichen relativen Werte wie für Umgebung 1 erzielt, vergleiche

Tabelle 5.4: Umgebung 2, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten im Vergleich zum Standard-SARSA-Algorithmus für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	t-Test LP	SZ	t-Test SZ
1	3.7%	0	48.35%	1
1.3	31.16%	1	61.22%	1
1.5	36.79%	1	62.3%	1
2	31.48%	1	52.83%	1
3	-6.11%	0	-0.88%	0

die beiden Tabellen 5.3 und 5.5. Die besten Ergebnisse in beiden Messkriterien liegen für den Wert $Skalierung = 2.0$ vor, gleich wie in Umgebung 1, und bedeuten eine signifikante Verbesserung des Lernfortschritts um 31.25% und der Stoßzahl um 59.37%. Für $Skalierung = 1.5$ in Umgebung 3 weist das LP-Kriterium eine Verbesserung von 26.62% auf, für Umgebung 1 liegt dieser Wert bei 25.54%. Die Stoßzahl verbessert sich um 54.9% in Umgebung 3 und um 57.17% in Umgebung 1. Beide Szenarien, Umgebung 1 und Umgebung 3, beinhalten keine engen Pässe, sind aber unterschiedlich aufgebaut.
Für die Umgebung BC werden nur die Werte $Skalierung = \{1.0, 2.0\}$ untersucht. Dies liegt an der

Tabelle 5.5: Umgebung 3, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten im Vergleich zum Standard-SARSA-Algorithmus für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	t-Test LP	SZ	t-Test SZ
1	20.32%	1	52.35%	1
1.3	27.93%	1	58.04%	1
1.5	26.62%	1	54.9%	1
2	31.25%	1	59.37%	1

Komplexität der Umgebung und der langen Zeit, die jeder Versuch benötigt. Für beide Werte des Parameters $Skalierung$ liegt keine signifikante Verbesserung des LP-Kriteriums vor, siehe Tabelle 5.6. Hingegen ist die Verbesserung der Stoßzahl gravierend und liegt bei über 60% für beide Werte. Die Umgebung BC beinhaltet auch einen engen Pass, siehe Raum Küche. Der Agent muss sehr fein navigieren, um durch diesen engen Pass zu kommen. Es wird vermutet, dass die gewählten Werte für den Parameter $Skalierung$ für den engen Pass nicht fein genug eingestellt sind, deswegen wird keine signifikante Verbesserung des LP-Kriteriums erzielt. Je größer (räumlich) die Umgebung ist, desto kleiner ist die

Tabelle 5.6: Umgebung BC, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten im Vergleich zum Standard-SARSA-Algorithmus für unterschiedliche Werte des Parameters $Skalierung$, der die Stärke der Steuerbefehle beeinflusst.

Skalierung	LP	t-Test LP	SZ	t-Test SZ
1	9.78%	0	60.815%	1
2	5.73%	0	63.45%	1

positive Auswirkung der eingebundenen reaktiven Fähigkeit auf die Lerngeschwindigkeit des Lernprozesses, vergleiche die Verbesserungen des Lernfortschritts für den Wert $Skalierung = 2.0$ in Umgebung 1 (31.98%, siehe Tabelle 5.3) mit Umgebung BC (5.73%, siehe Tabelle 5.6). Diese Tatsache lässt sich mit den langen Entscheidungsketten, die in größeren Umgebungen erlernt werden sollen, erklären. Die reaktive Fähigkeit vermeidet nicht alle Kollisionen, so dass es in längeren Entscheidungsketten mit großer Wahrscheinlichkeit zu einer Unterbrechung kommmt. Wenn eine Kollision stattfindet, wird der Agent an seinen Startpunkt zurückgesetzt. Es wird vermutet, dass aus diesem Grund die positive Auswirkung der Einbindung der reaktiven Fähigkeit in den Lernprozess mit der Größe der Umgebung nachlässt.
Die reaktive Fähigkeit erfüllt ihren Zweck, sie beschützt den Agenten vor Kollisionen mit einer Wand. Dabei kann dieser Schutzmechanismus auch sehr bedeutsam zur Beschleunigung des Lernprozesses beitragen. Die Beschleunigung des Lernprozesses hängt aber von der Beschaffenheit der verwendeten Umgebung ab.
Ein guter Schätzwert des Parameters $Skalierung$ für das verwendete neuronale Netz ist der Wert 1.5.

Wenn die entsprechende Zeit für die Kalibrierung des RL-Agenten vorhanden ist, wird empfohlen unterschiedliche Werte für den Parameter $Skalierung$ aus dem Intervall $[1; 2]$ zu untersuchen.

5.4.4 Optimale Parametrisierung des Vorinitialisierungsschrittes

Wenn ein ungenaues Umgebungsmodell zur Verfügung steht, kann die Q-Funktion durch vorhandene Informationen vorinitialisiert werden. Im Algorithmus 5.3 ist der Projektionsschritt des Kraftfelds auf die Q-Funktion dargestellt. Im entwickelten Algorithmus sind zwei Parameter p, n eingeführt, um den Vorinitialisierungsschritt entsprechend dem verwendeten Belohnungsmodell kalibrieren zu können, vergleiche Zeile 13 und 15. Im vorhandenen Kapitel werden die optimalen Parameter (p, n) für den eingeführten Vorinitialisierungsschritt in unterschiedlichen Umgebungen gesucht.
Ein Parameter $n < 0$ drückt für den Algorithmus 5.3 aus, wie stark eine Aktion, die nicht mit dem Vorschlag des Kraftfelds übereinstimmt, bestraft wird. Der Parameter $p > 0$ drückt aus, wie stark eine Aktion, die mit dem Vorschlag des Kraftfelds übereinstimmt, belohnt wird. Für den SARSA-Algorithmus soll die Relation zwischen den gewählten Parametern (p, n) und einem vorgegebenen Belohnungsmodell stimmen.
Die reaktive Fähigkeit des RL-Agenten hat eine ähnliche Auswirkung wie das erzeugte, abstoßende Kraftfeld. Der Vorinitialisierungsschritt wird ohne Einbindung der reaktiven Fähigkeit untersucht. So kann der Einfluss des Initialisierungsschrittes besser analysiert werden. Die Auswirkung des Vorinitialisierungsschrittes auf den RL-Agenten wird im nächsten Abschnitt analysiert.
In Tabelle 5.7 sind die in Umgebung 1 erzielten Ergebnisse für den um einen Vorinitialisierungsschritt erweiterten Standard-SARSA-Algorithmus mit unterschiedlichen Parametersätzen

$$(p, n) \in \{(+1, -5), (+0.1, -5), (+1, -2), (+0.1, -0.1)\}$$

zusammengefasst. Für die drei Parametersätze $\{(+1, -5), (+0.1, -5), (+1, -2)\}$ wird eine signifikante Verbesserung beider Messkriterien gemessen, siehe Zeilen 1 bis 3 in Tabelle 5.7. Für den Parametersatz $(+0.1, -0.1)$ werden keine signifikanten Unterschiede zum Standard-SARSA-Algorithmus für beide Kriterien festgestellt.
Eine Korrelation zwischen den beiden Messkriterien kann für alle Parametersätze festgestellt werden,

Tabelle 5.7: Umgebung 1, relative Verbesserung des Lernfortschritts und der Stoßzahl des um einen Vorinitialisierungsschritt erweiterten Standard-SARSA-Algorithmus im Vergleich zum nicht-erweiterten Standard-SARSA-Algorithmus, unterschiedlicher Parametersatz (p, n)

(p, n)	LP	t-Test LP	SZ	t-Test SZ
+1/-5	25.30%	1	25.65%	1
+0.1/-5	20.93%	1	22.3%	1
+1/-2	17.53%	1	15.75%	1
+0.1/-0.1	-0.86%	0	-3.13%	0

siehe Spalte 2 und 4 in Tabelle 5.7. Diese Korrelation beider Messkriterien wird für Umgebung 3 bestätigt, siehe Spalte 2 und 4 in Tabelle 5.9.
Für Umgebung 2 wird dagegen eine solche Abhängigkeit nicht erkannt, siehe Tabelle 5.8. Dies lässt sich durch die Beschaffenheit dieser Umgebung erklären, in der die Fähigkeit zum feinen Manövrieren gefragt ist. Für Umgebung 2 entstehen im Kraftfeld viele lokale Extrempunkte, so dass der Agent viele Schritte zum Umlernen benötigt, während des Umlernens und der vielen ausgeführten Schritte können

auch mehr Fehler, die in Kollisionen resultieren, entstehen.
Eine signifikante Verbesserung beider Kriterien wird in Umgebung 2 nur für die zwei Parametersätze

Tabelle 5.8: Umgebung 2, relative Verbesserung des Lernfortschritts und der Stoßzahl des um einen Vorinitialisierungsschritt erweiterten Standard-SARSA-Algorithmus im Vergleich zum Standard-SARSA-Algorithmus, unterschiedlicher Parametersatz (p, n)

(p, n)	LP	t-Test LP	SZ	t-Test SZ
+1/-5	2.55%	0	10.99%	1
+0.1/-5	9.73%	1	16.78%	1
+1/-2	7.09%	1	10.7%	1
+0.1/-0.1	-8.98%	1	-3.52%	0

$\{(+0.1, -5), (+1, -2)\}$ gemessen, siehe Zeile 2 und 3 in Tabelle 5.8. Diese Verbesserung ist kleiner als in Umgebung 1 und beträgt bis zu 10% beim Lernfortschritt und fast 17% bei der Stoßzahl für den Parametersatz $(+0.1, -5)$, siehe 2. Zeile, 2. und 4. Spalte in Tabelle 5.8. Diese Verbesserung wird mit einem Parametersatz erzielt, der sich vom besten Parametersatz $(+1, -5)$ aus Umgebung 1 unterscheidet. Mit diesem Parametersatz wird in Umgebung 2 eine geringe Verbesserung des Lernfortschritts gemessen, so dass kein signifikanter Unterschied zum Standard-SARSA-Algorithmus festgestellt wird, siehe Zeile 1, Spalte 2 und 3 in Tabelle 5.8. Dagegen weist die Stoßzahl bei der durch Vorwissen erweiterten Methode mit dem Parametersatz $(+1, -5)$ eine signifikante Verbesserung von fast $+11\%$ auf, siehe Zeile 1, Spalte 4 und 5 in Tabelle 5.8. Kleine Belohnungen für die Ausführung der Empfehlung des Kraftfeldes weisen bessere Ergebnisse im Messkriterium SZ auf.
Der nächste auffällige Punkt ist, dass der Parametersatz $(+0.1, -0.1)$ eine signifikante Verschlechterung des Lernfortschritts ergibt, siehe Spalten 2 und 3. Bei der Stoßzahl ist dagegen keine signifikante Verschlechterung zu beobachten, siehe Spalten 4 und 5 in Tabelle 5.8. Für Umgebung 2 wird gemessen, dass die Initialisierung der Q-Funktion mit Informationen aus dem Kraftfeld nicht so einen hohen Nutzen liefert wie erwartet. Dies lässt sich durch das Entstehen vieler lokaler Extrempunkte in den Kraftfeldern erklären, die dann umgelernt werden müssen.
In Umgebung 3 werden ähnliche Ergebnisse wie in Umgebung 1 erzielt. Hier liefern wieder drei Parametersätze $\{(+1, -5), (+0.1, -5), (+1, -2)\}$ eine signifikante Verbesserung von bis zu 30% bei beiden Kriterien. Für den Parametersatz $(+0.1, -0.1)$ wird kein signifikanter Unterschied zum Standard-SARSA-Algorithmus festgestellt, siehe 4. Spalte in Tabelle 5.9. In dieser Umgebung ist der beste Parametersatz wie in Umgebung 1 $(+1, -5)$. Der Lernfortschritt weist eine Verbesserung von 29.5% auf, die Stoßzahl verbessert sich um 29.82%, siehe Zeile 1 in Tabelle 5.9. Der Parametersatz $(+0.1, -5)$ liefert aber sehr ähnliche Ergebnisse, wie der Parametersatz $(+1, -5)$, der Lernfortschritt verbessert sich um 29.1% und die Stoßzahl um 28.59%. In Umgebung 3 kann die Frage nach dem optimalen Parametersatz nicht so eindeutig beantwortet werden, wie in Umgebung 1.
In der komplexeren Umgebung BC werden die in Umgebung 1 getroffenen Aussagen für den Parametersatz $(+0.1, -5)$ bestätigt, siehe Tabelle 5.10. Die relative Verbesserung ähnelt der Verbesserung in Umgebung 1. Eine Korrelation zwischen beiden Kriterien wird festgestellt.
Die Einbindung der zusätzlichen Informationen kann den Lernprozess beschleunigen und die Anzahl der Kollisionen des Agenten mit Wänden verringern. Eine korrekte Relation zwischen dem Belohnungsmodell und der Parametrisierung dieses Schrittes ist von großer Bedeutung. Wenn die richtige Relation bestimmt worden ist, wird eine Beschleunigung des Lernprozesses und eine Verringerung der Kollisionen erzielt. Bei den meisten Versuchen wird eine Korrelation zwischen der Verbesserung beider Messkriterien beobachtet.

Tabelle 5.9: Umgebung 3, relative Verbesserung des Lernfortschritts und der Stoßzahl des um einen Vorinitialisierungsschritt erweiterten Standard-SARSA-Algorithmus im Vergleich zum Standard-SARSA-Algorithmus, unterschiedlicher Parametersatz (p, n)

(p, n)	LP	t-Test LP	SZ	t-Test SZ
+1/-5	29.5%	1	29.82%	1
+0.1/-5	29.1%	1	28.59%	1
+1/-2	23%	1	19.61%	1
+0.1/-0.1	0.38%	0	0.76%	0

Tabelle 5.10: Umgebung BC, relative Verbesserung des Lernfortschritts und der Stoßzahl des um einen Vorinitialisierungsschritt erweiterten Standard-SARSA-Algorithmus im Vergleich zum Standard-SARSA-Algorithmus, unterschiedlicher Parametersatz (p, n)

(p, n)	LP	t-Test LP	SZ	t-Test SZ
+0.1/-5	20.86%	1	22.44%	1

Die Verbesserung des Lernfortschritts liegt zwischen fast 10% für Umgebung 2 und fast 30% in Umgebung 3 für den besten Parametersatz. Der Unterschied zwischen den erzielten Ergebnissen für den Lernfortschritt von bis zu 20 Prozentpunkten lässt die Vermutung zu, dass eine Abhängigkeit der Verbesserung durch die Einbindung des Vorinitialisierungsschrittes von der Beschaffenheit der Umgebung existiert. Diese Abhängigkeit kann durch die Entstehung einer unterschiedlichen Anzahl an lokalen Extrempunkten in den erzeugten Kraftfeldern für unterschiedliche Umgebungen erklärt werden.
An dieser Stelle soll auch bemerkt werden, dass die für eine Vorinitialisierung benötigten Informationen eine sehr starke Bedingung darstellen und mit Planungsalgorithmen besser genutzt werden könnten. Der Vorteil liegt aber in der Fähigkeit des RL-Agenten das ungenaue Wissen zu adaptieren, was die gewöhnlichen Planungsalgorithmen ohne Erweiterungsschritte nicht erlauben. Die Qualität der angewendeten Informationen steht in einer schlechten Relation zur erzielten Verbesserung von nicht mehr als 30% unter Voraussetzung einer guten Kalibrierung.

5.4.5 Kombination der möglichen Erweiterungen

Die unterschiedlichen Kombinationen der einzelnen Erweiterungen und deren Auswirkung auf den Lernprozess werden im Folgenden analysiert. Diese Kombinationen werden nur in Umgebung 3 untersucht. Dabei wird die in vorherigen Kapiteln erzielte, optimale Parametrisierung des RL-Agenten angewendet. Die Umgebung 3 wird wegen ihrer mittleren Größe und ihrem mittleren Schwierigkeitsgrad beim Lernen[34] ausgewählt.
Folgende mögliche Kombinationen werden untersucht:

- RL-Agent mit SARSA*-Algorithmus, siehe Gleichung 4.2
- RL-Agent mit Vorinitialisierungsschritt aus Kapitel 5.3

Die erste Kombination, die hier behandelt wird, ist die Einbindung des erweiterten Lernschrittes, siehe Gleichung (4.2) in den RL-Agenten. Der Parameter $Skalierung$ wird für den RL-Agenten auf den Wert 1.3 gesetzt.
Die erzielten Ergebnisse sind für zwei unterschiedliche Werte des heuristischen Einflussparameters k in

[34] Umgebung 3 besitzt eine mittlere Anzahl an Zuständen und keine engen Pässe.

Tabelle 5.11 zusammengestellt. Eine signifikante Verbesserung des Lernfortschritts von bis zu 40% im Vergleich zum Standard-SARSA-Algorithmus wird gemessen.
Die einzelnen Erweiterungen versprechen aber schon einer Verbesserung. Für den RL-Agenten wird eine Verbesserung im LP um bis zu 27.93% für $Skalierung = 1.3$ erzielt, siehe Tabelle 5.5. Bei Anwendung der heuristischen Funktion im Standard-SARSA-Algorithmus ($k = -0.5$) wird eine Verbesserung des LP um bis zu 23.57% erzielt, siehe Tabelle 4.10. Mit der Kombination, in der der RL-Agent den SARSA*-Algorithmus ($k = -0.5, Skalierung = 1.3$) anwendet, wird eine Verbesserung von bis zu 39.52% erzielt, siehe zweite Zeile in Tabelle 5.11.
Wenn nur der SARSA*-Algorithmus eingebunden wird, ist die Stoßzahl mit einer relativen Änderung von -1.51% nicht signifikant schlechter, vergleiche zweite Zeile in Tabelle 4.10. Diese Auswirkung der heuristischen Funktion bleibt auch hier bestehen. Die Verbesserung der Stoßzahl für den RL-Agenten ohne Einfluss der heuristischen Funktion mit dem Parameter $Skalierung = 1.3$ liegt bei 58.04%, siehe zweite Zeile in Tabelle 5.5. Für den gleich konfigurierten RL-Agenten, der den SARSA*-Algorithmus anwendet, liegt für das Messkriterium Stoßzahl eine ähnlich signifikante Verbesserung von 57.78% für $k = -0.3$ vor, bzw. 55.38% für $k = -0.5$. Ein Unterschied von etwa 2.4 Prozentpunkten bleibt bestehen. In manchen Situationen entscheidet sich der Agent als Folge der heuristischen Funktion für eine gefährlichere Aktion, die den Agenten zu nah an die Wand steuert.
Die Anwendung des SARSA*-Algorithmus im RL-Agenten weist eine geringe negative Auswirkung auf die Stoßzahl auf, hingegen übt sie eine deutlich positive Auswirkung auf den Lernfortschritt aus.
Die Anwendung des SARSA*-Algorithmus im RL-Agenten liefert eine Verbesserung um bis zu 40%,

Tabelle 5.11: Umgebung 3, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten bei Anwendung des SARSA*-Algorithmus im Vergleich zum Standard-SARSA-Algorithmus, Parameter $Skalierung := 1.3$, unterschiedliche Parametrisierung des heuristischen Einflussparameters k

k	LP	t-Test LP	SZ	t-Test SZ
-0.3	40.41%	1	57.78%	1
-0.5	39.52%	1	55.38%	1

benötigt nicht viel Vorbereitung bei der Realisierung (Erweiterung des Belohnungssignals und Einbindung der reaktiven Fähigkeit, falls vorhanden) und kann empfehlenswert sein.
Die zweite Kombination, RL-Agent mit Vorinitialisierungsschritt, erzielt eine Verbesserung des Lernfortschritts im Vergleich zur Standard-SARSA-Methode um etwa 45% und eine Verringerung der Anzahl an Kollisionen von knapp 70%, siehe Tabelle 5.12. Die Vorinitialisierung der Q-Funktion und die Anwendung der reaktiven Fähigkeit in Gefahrensituationen haben ein ähnliches Ziel, den Agenten weg von der Wand zu steuern. Damit wird in kritischen Situationen doppelt aufgepasst.

Tabelle 5.12: Umgebung 3, relative Verbesserung des Lernfortschritts und der Stoßzahl des RL-Agenten mit Vorinitialisierungsschritt im Vergleich zum Standard-SARSA-Algorithmus, Parameter $Skalierung := 1.3$

(p, n)	LP	t-Test LP	SZ	t-Test SZ
+0.1/-5	45.31%	1	69.16%	1

5.5 Zusammenfassung

Der Vorteil des entwickelten RL-Agenten liegt in der Einfachheit der Implementierung. Der entwickelte RL-Agent ist im Onlinebetrieb anwendbar und benötigt keine komplizierte Architektur. Der Lernvorgang ist effizienter als der Lernvorgang mit gewöhnlichem SARSA-Algorithmus, ist aber immer noch nicht effizient genug, um in Umgebungen mit großem Bewegungsareal Anwendung zu finden. Der um reaktive Fähigkeiten erweiterte SARSA-Algorithmus beinhaltet ein paar Parameter, die richtig kalibriert werden müssen. Für die beste Wirkung der vorgenommenen Erweiterung um eine reaktive Fähigkeit wird der Kalibrierschritt für den RL-Agenten benötigt.
Das für das Erstellen der Vorinitialisierung des Parametersets (p, n) benötigte Wissen stellt eine sehr hohe Anforderung dar, kann nur in seltenen Fällen zur Geltung kommen und könnte wahrscheinlich viel effizienter angewendet werden, wenn Planungsalgorithmen in Anspruch genommen werden.

Kapitel 6

Bestimmung von Aktionen der höheren Abstraktionsebene

Die Aktionen der höheren Abstraktionsebene werden aus reaktiven Fähigkeiten gebildet. Die generierten reaktiven Fähigkeiten sollen eine gewisse Autonomie aufweisen. Sie sollen also mehrere Zeitschritte ohne Betätigung des Leistungselements ausgeführt werden können, und sie werten nur die aktuelle Situation aus. Für die Generierung der benötigten reaktiven Fähigkeiten werden die unterschiedlichen Sensoren des Scorpion-Roboters angewendet. Die reaktiven Fähigkeiten werden in der vorliegenden Arbeit zuerst auf Grundlage der verarbeiteten Sensorinformationen voneinander unterschieden.
Die erste Art dieser reaktiven Fähigkeiten verarbeitet die globalen Informationen über die aktuelle Position und das verfolgte Ziel in den auszuführenden Befehlen der Motorsteuerung. Die Realisierung der reaktiven Fähigkeit erster Art wird mit einem P-Regler implementiert und repräsentiert die reaktive Fähigkeit ”zum Ziel fahren”. Diese reaktive Fähigkeit wird im Agenten mit Doppelpass-Architektur und Reinforcement-Learning, dem sogenannten DPA-RL-Agenten, durch Vorgabe des Ziels als eine abstrakte Aktion definiert. Die Realisierungen der reaktiven Fähigkeit ”zum Ziel fahren” unterscheiden sich nach vorgegebenen globalen Informationen. Die globalen Informationen können aus unterschiedlichen Quellen gewonnen werden, z. B. aus Bildinformationen oder aus Odometriedaten. In diesem Kapitel werden die möglichen Realisierungen dieser reaktiven Fähigkeit beschrieben.
Die zweite Art der reaktiven Fähigkeiten verarbeitet lokale Informationen aus IR-Sensoren. Diese Art repräsentiert die reaktive Fähigkeit des Agenten ”Hindernis umfahren” und wandelt die unbehandelten Abstandsmessungen der IR-Sensoren in Befehle für die Motoren des Scorpion-Roboters um.
Je nach Anwendung und verfolgter Aufgabe kann diese zweite Art der reaktiven Fähigkeit in zwei weitere Klassen unterteilt werden.
Für den RL-Agenten hat die reaktive Fähigkeit die Aufgabe eines Schutzmechanismus. In diesem Fall ist es wichtig so schnell wie möglich den Agenten aus der Gefahrensituation zu steuern. Somit entsteht die reaktive Fähigkeit ”Hindernis umfahren”.
Die Aufgabe der im DPA-RL-Agenten angewendeten, reaktiven Fähigkeit besteht darin, mit einer vorgegebenen Richtung ein Hindernis zu umfahren. Die vom DPA-RL-Agenten benötigten, reaktiven Fähigkeiten unterscheiden sich in der vorgegebenen Richtung, ”Hindernis rechts/links umfahren”. Links umfahren bedeutet im Uhrzeigersinn und rechts umfahren bedeutet gegen den Uhrzeigersinn. Die beiden reaktiven Fähigkeiten sind symmetrisch definiert. Die beiden Möglichkeiten bilden zwei abstrakte Aktionen, die im DPA-RL-Agenten angewendet werden.
In Kapitel 6.1 werden zuerst wichtige Begriffe wie reaktive Fähigkeit und abstrakte Aktion eingeführt und anhand von Beispielen erläutert. Die in der vorliegenden Arbeit verwendeten reaktiven Fähigkeiten werden eingeführt. Die erste Art der reaktiven Fähigkeit ”zum Ziel fahren”, die globale Sensorinformationen verarbeitet, wird in diesem Kapitel genauer spezifiziert.
Für das Erzielen der reaktiven Fähigkeit ”Hindernis umfahren” bzw. ”Hindernis rechts umfahren” und

”Hindernis links umfahren” werden zwei unterschiedliche Ansätze eingeführt. Dabei werden für diese Aufgabenstellung sowohl konventionelle Methoden der Regelungstechnik als auch nicht konventionelle Methoden wie der evolutionäre Ansatz eingesetzt und die erzielten Ergebnisse miteinander verglichen. Es existiert die Möglichkeit die reaktiven Fähigkeiten im Onlinebetrieb zu erzielen. Diese Möglichkeit kann mit der Reinforcement-Learning-Methode realisiert werden, siehe z.B. [Papierok et al., 2008] und [Papierok, 2007].
In Kapitel 6.2 wird das Erzielen der reaktiven Fähigkeit zweiter Art mit dem evolutionären Ansatz bei Verarbeitung von lokalen Sensorinformationen eingeführt.
In Kapitel 6.3 wird die Möglichkeit, reaktive Fähigkeiten zweiter Art mit konventionellen Methoden der Regelungstechnik zu erzeugen, eingeführt.
Die beiden Arten der Realisierung von ”Hindernis umfahren” werden in Kapitel 6.4 miteinander verglichen.

6.1 Reaktive Fähigkeiten und darauf basierende abstrakte Aktionen

6.1.1 Begriffsklärung

Definition 6.1.1 (Kontrollfunktion). *Als Kontrollfunktion wird die direkte Abbildung von Sensorwerten (Daten aus Bildern, Odometriedaten, IR-Sensordaten) auf Steuerbefehle des Antriebssystems des Roboters (ω, v) bezeichnet, wobei mit v die Vorwärtsgeschwindigkeit und mit ω die Drehgeschwindigkeit bezeichnet wird.*

Definition 6.1.2 (reaktive Fähigkeit). *Eine reaktive Fähigkeit ist eine Fähigkeit des Agenten, sich auf Grundlage der aktuellen Sensorinformationen autonom zu bewegen. Die reaktive Fähigkeit soll eine gewisse Autonomie aufweisen und ist durch die verwendeten Sensoren und verfolgten Ziele geprägt.*

Die reaktive Fähigkeit kann erstens aus mehreren Teilschritten, die als Kontrollfunktionen realisiert sind, bestehen, zweitens berechnet sie selbst in Abhängigkeit von der aktuellen Situation den möglichen Befehl auf die Motorsteuerung, und drittens besitzt sie keine planende Eigenschaften.
Ein Beispiel für die reaktive Fähigkeit ist die Fähigkeit des Agenten zu einem Ziel zu fahren. Diese reaktive Fähigkeit ist noch nicht spezifiziert und bezeichnet die Fertigkeit des Agenten sich vom aktuellen Punkt zum vorgegebenen Punkt bzw. Objekt zu bewegen.

Definition 6.1.3 (abstrakte Aktion). *Eine abstrakte Aktion wird durch eine konkretisierte reaktive Fähigkeit repräsentiert. Die abstrakte Aktion hat die gleichen Fertigkeiten wie die reaktive Fähigkeit auf deren Grundlage diese abstrakte Aktion basiert.*

Zum Beispiel wird durch Vorgabe eines Ziels ”Küche” in der reaktiven Fähigkeit ”fahre zum Ziel” die abstrakte Aktion ”fahre in die Küche” konstruiert. Durch Vorgabe eines anderen Ziels in der reaktiven Fähigkeit ”zum Ziel fahren”, zum Beispiel ”Büro Nr. 3”, wird die nächste abstrakte Aktion konstruiert, ”fahre ins Büro Nr. 3”. Und der Agent beinhaltet in seinem Aktionsraum zwei unterschiedliche abstrakte Aktionen ”fahre in die Küche” und ”fahre ins Büro Nr. 3”, die auf derselben reaktiven Fähigkeit basieren. Eine abstrakte Aktion kann mit der reaktiven Fähigkeit identisch sein. Die reaktive Fähigkeit *Hindernis umfahren* ist gleich mit der abstrakten Aktion, die diese Fähigkeit besitzt. Abstrakte Aktionen werden zum Beispiel im DPA-RL-Agenten verwendet.

Dieser Begriff kommt dann ins Spiel, wenn mehrere mögliche abstrakte Aktionen durch die Konkretisierung der reaktiven Fähigkeiten vorgegeben sind, wenn das Markowsche Entscheidungsproblem vordefiniert ist, und wenn der Agent eine Kette aus Aktionen erlernt, um das vorgegebene Problem zu lösen.

6.1.2 Reaktive Fähigkeit "zum Ziel fahren"

Die von Agenten benötigte reaktive Fähigkeit "zum Ziel fahren" verarbeitet die globalen Informationen über die aktuelle Position und das verfolgte Ziel in den auszuführenden Befehlen der Motorsteuerung, ohne Berücksichtigung der Gegebenheiten der Umgebung.

$$(\omega, v) \leftarrow g(e_w(t), e_v(t))$$

Für die Realisierung dieser Fähigkeit wird ein P-Regler verwendet, der den Agenten immer zum Ziel steuert, wenn das Ziel vorgegeben ist. Zwei mögliche Realisierungen werden beschrieben, die sich in ihren Inputdaten unterscheiden: globale Daten über die aktuelle Position des Agenten und die Zielposition oder aus Bildern abgeleitete Informationen über die Zielposition.
Die globalen Informationen werden vom entwickelten Regler in Kontrollbefehle auf die verwendete Motorsteuerung des Agenten umgerechnet. Die Motoren des Roboters werden durch zwei vorgegebene Geschwindigkeiten gesteuert, die Drehgeschwindigkeit und die Vorwärtsgeschwindigkeit (ω, v). Beide Geschwindigkeiten werden durch einen P-Regler bestimmt, siehe Gleichung (6.1) und (6.2).

$$\omega(t) = K_w \cdot e_w(t) \propto (\alpha^{soll} - \alpha^{ist}) \tag{6.1}$$

$$v(t) = K_v \cdot e_v(t) \propto ||P^{soll} - P^{ist}|| \tag{6.2}$$

Zuerst wird der Regler vorgestellt, der die benötigten Informationen in Form von globalen Informationen aus Odometriedaten extrahiert. Durch die verwendeten Odometrie-Sensoren und dem darauf basierenden Kompass-Sinn-Sensor, ist der Agent immer in der Lage die Zielrichtung bezüglich seines lokalen Koordinatensystems zu bestimmen. Die Differenz zwischen der Soll-Orientierung α^{soll}, die mit dem Kompass-Sinn bestimmt wird, und der Ist-Orientierung α^{ist}, die aus den Odometriedaten gewonnen wird, dient als Grundlage für die Regelung der Orientierung des Roboters, siehe Abbildung 5.3.

$$e_w(t) = (\alpha^{soll}(t) - \alpha^{ist}(t))$$

Je nach verfolgtem Ziel kann die Vorwärtsgeschwindigkeit des Agenten entweder auf einen konstanten Wert gesetzt werden oder mit dem Regler gesteuert werden[35]. Im Fall mit konstanter Geschwindigkeit $e_v(t) = const$ darf dieser Wert nicht zu hoch gesetzt sein, da die Gefahr besteht, dass die Zielposition verpasst wird. Wenn die Geschwindigkeit in Abhängigkeit vom Abstand zum Ziel geregelt wird, sieht der Fehler für den Regler wie folgt aus:

$$e_v(t) = ||P^{soll}(t) - P^{ist}(t)||$$

wobei die Werte $P^{soll}(t)$ und $P^{ist}(t)$ die Position des Ziels und die Position des Agenten repräsentieren. Die Positionen P^{ist} und P^{soll} sollen im verwendeten Umgebungskoordinatensystem vorgegeben sein, z. B. in Bezug auf die Startposition des Agenten.
Die reaktive Fähigkeit "zum Ziel fahren" kann auf Grundlage der bildbasierten Informationen realisiert

[35] In der vorliegenden Arbeit wurden beide Varianten ausprobiert. Es wurde kein besonderer Vorteil einer der beiden Realisierungen festgestellt.

werden, siehe Abbildung 6.1. In diesem Fall wird die Methode der visuell basierten Regelung [Hutchinson et al., 1996] angewendet. Eine solche Realisierung ist unabhängig von Odometriedaten und auch von den durch Odometriedaten entstehenden Fehlern, ist aber anfällig für Fehler, die von den Bildverarbeitungsmethoden und bei der Bildaufnahme verursacht werden. Eine wichtige Bedingung für die Anwendung dieser Methode ist, dass das Zielobjekt im Bild sein muss und sicher erkannt werden muss. Somit gibt die reaktive Fähigkeit "zum Ziel fahren" zwei Befehle auf die Motoren $(\omega(t), v(t))$ als Ant-

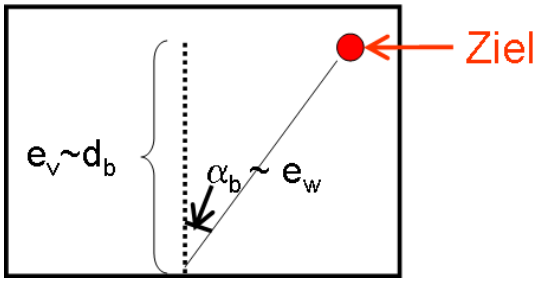


Abbildung 6.1: Angewendete Bildinformationen

wort auf die aus dem Bild abgeleiteten Informationen α_b, d_b aus.
Der angewendete Regler muss entsprechend des verfolgten Zieles kalibriert sein, die konstanten Werte K_w, K_v müssen richtig bestimmt werden.

6.1.3 Reaktive Fähigkeit "Hindernis (rechts/links) umfahren"

Die nächste Art der reaktiven Fähigkeiten verarbeitet Informationen über die lokale Umgebung in Steuerbefehle und repräsentiert die reaktive Fähigkeit "Hindernis umfahren". In diesem Fall ist die reaktive Fähigkeit gleich der abstrakten Aktion. Je nach Anwendung sind die Aufgaben dieser reaktiven Fähigkeit unterschiedlich. Im RL-Agenten ist die Aufgabe, den Agenten zu schützen, und im DPA-RL-Agenten, je nach vorgegebener Richtung entlang des Hindernisses zu fahren.
Die verarbeiteten Sensorinformationen der Scorpion-Roboter sind ungenau, siehe zum Beispiel [Köpsel, 2007]. Um das Problem mit Ungenauigkeiten zu umgehen, wird eine realitätsnahe Simulation, siehe auch Kapitel 2.6, zum Erzielen dieser reaktiven Fähigkeiten eingesetzt. Die im Offline-Modus erzielten lokalen Sensorinformationen werden zur Bestimmung der gewünschten reaktiven Fähigkeiten verwendet.
Die Realisierung der reaktiven Fähigkeit kann als Kontrollfunktion $f(s_1, \ldots, s_7)$ dargestellt werden:

$$(\omega, v) \leftarrow f(s_1, \ldots, s_7)$$

wobei die Werte $s_1, \ldots, s_7$ von sieben gewählten IR-Sensoren des Scorpion-Roboters geliefert werden, siehe Abbildung 2.4. Die produzierten Werte sind die im Antriebssystem benötigten Befehle für die Motorsteuerung, siehe Kapitel 2.5.3. Diese Funktion kann direkt als eine Funktion realisiert werden oder aus mehreren Teilfunktionen zusammengesetzt werden. Die gewünschten reaktiven Fähigkeiten können durch Anwendung des evolutionären Ansatzes oder auch durch Anwendung der klassischen Regelungsmethoden erzielt werden. Mit dem evolutionären Ansatz wird ein neuronales Netz erzeugt, das die gewünschte Fähigkeit aufweist. Mit der klassischen Regelungsmethode wird die benötigte Funktion $f(s_1, \ldots, s_7)$ aus vier Teilschritten zusammengesetzt.

6.2 ”Hindernis umfahren” mit evolutionärem Ansatz

6.2.1 Grundidee

Die gesuchte Kontrollfunktion $(\omega, v) \leftarrow f(s_1, \ldots, s_7)$, die eine der benötigten, reaktiven Fähigkeiten ”Hindernis umfahren”, ”Hindernis rechts umfahren” oder ”Hindernis links umfahren” realisiert, kann durch ein künstliches neuronales Netz (kNN)

$$(\omega, v) \leftarrow kNN(s_1, \ldots, s_7) \approx f(s_1, \ldots, s_7)$$

repräsentiert werden. Als Eingabe dienen sieben, von den IR-Sensoren gemessene Werte $s_1, \ldots, s_7$. Als Ausgabesignal liefert die Funktion die benötigten Steuerbefehle auf die Motoren (ω, v). Die Eingabeschicht eines kNN enthält somit acht Knoten: sieben auf das Intervall $[0, 1]$ normierte Sensorwerte und einen Bias-Knoten. Nach der Propagierung gibt das kNN zwei Befehle für die Motorsteuerung aus. Jedes Netz hat zwei Ausgangsknoten in der Ausgabeschicht.
Zur Erzielung des kNN mit gewünschten reaktiven Fähigkeiten wird beispielsweise der Ansatz *Neuro Evolution of Augmenting Topologies* (NEAT) von Stanley verwendet [Stanley & Miikkulainen, 2002]. NEAT wird wegen seiner Anpassungsfähigkeit und relativen Einfachheit in der Anwendung ausgewählt. Dieser evolutionäre Ansatz bestimmt sowohl die Topologieentwicklung als auch die Gewichte eines kNN. Bei der Anwendung der NEAT-Methode werden künstliche neuronale Netze erzeugt, die die gewünschten Fähigkeiten, sich entsprechend der Situation zu drehen und am Hindernis entlang zu fahren oder sich so schnell wie möglich vom Hindernis wegzudrehen, gut oder weniger gut beherrschen. In der vorliegenden Arbeit wird das von Mat Buckland [Buckland, 2002] implementierte Paket Windows NEAT verwendet.
Das benötigte kNN wird durch Anwendung des evolutionären Ansatzes bestimmt. Das grundlegende Prinzip lautet: Der Stärkere überlebt. Dabei wird eine ganze Population möglicher kNN-Individuen erzeugt. Die kNN-Individuen werden in den konstruierten Szenarien eingesetzt. Ihre Tauglichkeit zum Erzielen der gewünschten Fähigkeit wird mithilfe einer Fitnessfunktion gemessen. Je nachdem wie gut sich die einzelnen Individuen in den konstruierten Szenarien verhalten, werden aus besseren Individuen die Eltern für die neue Population selektiert. Mithilfe von Rekombinations- und Mutationsmechanismen wird eine neue Population von kNN’s aus ausgewählten Eltern erzeugt. Ähnliche Ideen werden auch in weiteren zahlreichen Literaturquellen verfolgt, unter anderem [Igel & Sendhoff, 2005], [Kassahun, 2006]. Einen guten Einstieg in die grundlegenden Begriffe und Mechanismen der evolutionären Algorithmen bietet zum Beispiel das Buch [Gerdes et al., 2004].
Die Einfachheit der Gewinnung und Anwendung der mithilfe evolutionärer Ansätze bestimmten kNN ist sehr vielversprechend. Zur Bestimmung werden nur Beispiel-Szenarien und eine Fitnessfunktion benötigt. Auch ein Nutzer ohne Programmierkenntnisse kann solche reaktiven Fähigkeiten erstellen, da nur eine intuitive Vorstellung von der Aufgabe des Agenten genügt.
Das mit evolutionären Algorithmen erzielte kNN benötigt im Gegensatz zu gewöhnlichen Regelkreisen keine aufwändige Kalibrierung, weist ein flexibleres Verhalten und eine gute Generalisierbarkeit auf und benötigt keine Filter für die Inputdaten. Eine Veränderung der Roboterkonfiguration im Fall der Anwendung der evolutionären Algorithmen (EA) ist weniger schmerzhaft, als beim konventionellen Regler-Ansatz.

6.2.2 Fitnessfunktion

Die Fitnessfunktion wird für eine intuitive Verständlichkeit so einfach wie möglich gewählt. Das erste Kriterium, mit dem die Güte der von einem Individuum gefahrenen Strecke gemessen werden kann, wird über den zurückgelegten Winkel in Bezug auf die bevorzugte Richtung (im Uhrzeigersinn, gegen den Uhrzeigersinn) definiert. Diese Art von Fitnessfunktion wird als Winkelfitness bezeichnet. Winkelfitness ist wichtig für die Bildung der kNN, die die im DPA-RL-Agenten verwendete reaktive Fähigkeit "Hindernis links/rechts umfahren" repräsentieren.
Der Fitnesswert der Winkelfitness wird aus dem zurückgelegten, in Abbildung 6.2 skizzierten Winkel

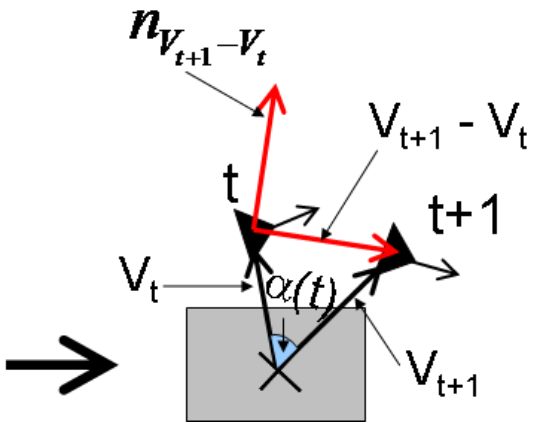


Abbildung 6.2: Zurückgelegter Winkel $\alpha(t)$ im Zeitabschnitt $[t, t+1]$. Der Roboter soll sich im Uhrzeigersinn bewegen, die gewünschte Richtung ist durch den Pfeil in der linken Ecke gekennzeichnet, $o^{soll} = 1$. Die wichtigen zur Prüfung der gewünschten Richtung angewendeten Vektoren sind $\mathbf{V}_t, \mathbf{V}_{t+1}, n_{V_{t+1}-V_t}$. Das Zentrum des Szenarios ist durch das Kreuz gekennzeichnet und wird zur Berechnung der benötigten Vektoren angewendet.

$\alpha(t)$ im Zeitabschnitt $[t, t+1]$ berechnet. Die gewünschte Richtung o^{soll} wird durch die Szenarien vorgegeben. $o^{soll} = 1$ bedeutet im Uhrzeigersinn, $o^{soll} = -1$ bedeutet gegen den Uhrzeigersinn. Das Zentrum des Szenarios, das in der Abbildung durch ein Kreuz gekennzeichnet ist, ist durch das Szenario vorgegeben. Der Agent bekommt eine Belohnung, wenn mindestens einer der Infrarotsensoren ein Hindernis spürt und er sich in die richtige, vom Szenario vorgegebene Richtung dreht, vergleiche Gleichung (6.3).

$$f^i_{t,Wink.}(s^i_{min}(t), \alpha^i(t)) = \begin{cases} \alpha^i(t) & : \quad s^i_{min}(t) < s_{max} \wedge (n^i_{V_{t+1}-V_t})^T \cdot V^i_{t+1} \cdot o^{soll} > 0 \\ 0 & : \quad \text{sonst} \end{cases} \tag{6.3}$$

wobei $\alpha^i(t)$ der vom Individuum i zurückgelegte Winkel im Zeitabschnitt $[t, t+1]$ ist, $s^i_{min}(t) = \min_{j=\{1,\ldots,7\}} s^i_j$ ist der kleinste Wert aus allen von IR-Sensoren zurückgegebenen Werten. Die Vektoren $V^i_t, V^i_{t+1}, n^i_{V_{t+1}-V_t}$ sind in Abbildung 6.2 dargestellt und werden zur Prüfung, ob sich der Agent in die vom Szenario vorgegebene Richtung o^{soll} bewegt, verwendet.
Das zweite Kriterium für die Fitnessfunktion wird basierend auf dem lokalen Verhalten des Agenten definiert. Die Idee zur Bestimmung dieser Definition der Fitnessfunktion stammt aus [Kassahun, 2006]. Die in der Funktion fließenden Informationen basieren nur auf lokalen Sensorwerten und lokalem Verhalten des Agenten, den Odometriedaten (Drehgeschwindigkeit $\omega^i(t)$ und Vorwärtsgeschwindigkeit $v^i(t)$ des Agenten) und den Sensorwerten $s^i_{min}(t) = \min_{j=\{1,\ldots,7\}} s^i_j$. Eine schnelle Drehung wird als "nicht gut"

angesehen, dagegen wird eine schnelle Fahrt ohne Drehung als "gut" angesehen. Dabei gilt die Regel: Je weiter weg sich das Objekt vom Roboter befindet, desto besser.

$$f^i_{t,Verhalten}(s^i_{min}(t), \omega^i(t), v^i(t)) = v^i(t) \cdot \exp^{-100\cdot(\omega^i(t))^2} \cdot s^i_{min}(t)$$

Der Fitnesswert f^i_t des Individuums i zu einem Zeitschritt t wird aus den beiden genannten Fitnesswerten zusammengefasst, wobei die Gewichtung der einzelnen Kriterien (Winkelfitness und Verhaltensfitness) deren Wichtigkeit für das gewünschte Ergebnis ausdrückt, also $f^i_t = w_{Winkel} \cdot f^i_{t,Wink.} + w_{Verhalten} \cdot f^i_{t,Verhalten}$. Die Gesamtfitness für das Individuum i und ein $Szenario_j$ wird aus den einzelnen Fitnesswerten $F^{Szenario_j}(i) = \sum_{t=0}^{T_j} f^{i,j}_t$ zusammengestellt, wobei durch T_j die für den Trainingsprozess vorgegebene Zeit in Szenario j bezeichnet wird. Dies wird für alle Szenarien durchgeführt, und der Fitnesswert wird aus allen Szenarien berechnet. Auf Grundlage der in allen Szenarien bestimmten Fitness wird ein Urteil gebildet, wie gut sich das durch einzelne kNN repräsentierte Individuum i in der aktuellen Episode verhalten hat.

6.2.3 Szenarien

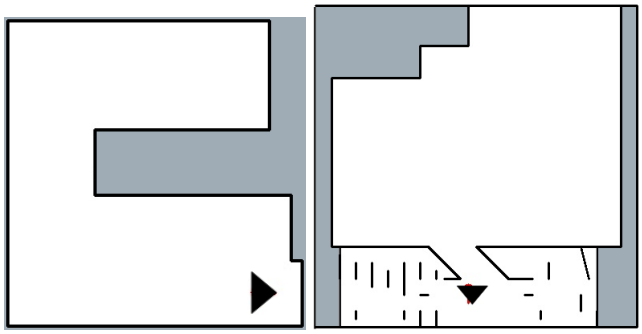
Durch jedes Szenario $j = 1, \ldots, \underbrace{|Szenarien|}_{\text{Anzahl Szenarien}}$ soll die vorgegebene Zeit T_j im Trainingsprozess, die Gewichtung der jeweiligen Fitnessanteile $w_{Winkel}, w_{Verhalten}$, die bevorzugte Richtung o^{soll} im Szenario sowie die Startposition, Startorientierung des Agenten und das Zentrum eines Szenarios vorgegeben werden. Die Gesamtfitness des Individuums i wird für jede Episode wie folgt berechnet: $F^{gesamt}(i) = \sum_{j=1}^{|Szenarien|} F^{Szenario_j}(i)$. Durch den Wert T_j wird die Wichtigkeit des Szenarios im Trainingsprozess vorgegeben. Wenn ein Individuum in einem Szenario viel Zeit zur Verfügung hat, kann es theoretisch viele Fitnesspunkte sammeln. Das Individuum bleibt stehen, wenn eine Kollision mit einem Hindernis stattgefunden hat, und wird automatisch bestraft, da seine Mitgenossen im Zeitabschnitt brav die Fitnesspunkte sammeln. Wenn aber nur wenig Zeit für ein Szenario vorgegeben ist, wirkt sich das nicht so dramatisch auf den gesamten Fitnesswert aus, als wenn es in einem Szenario erfolgte, in dem das Individuum viel Zeit zur Verfügung hat.

Zuerst werden die angewendeten Szenarien für das Erzielen der reaktiven Fähigkeit "Hindernis umfahren" beschrieben. Diese reaktive Fähigkeit wird im RL-Agent angewendet. Unwichtig ist dabei von welcher Seite der Agent das Hindernis umfährt. Es ist wichtiger, wie sich der Agent lokal verhält. In diesem Fall ist der Anteil der Fitnessfunktion, der das Verhalten des Agenten ($f_{t,Verhalten}$) beurteilt, von größerer Bedeutung.

Für das Erzielen der reaktiven Fähigkeit "Hindernis umfahren" werden insgesamt sechs Szenarien erstellt, $Szenario_0, \ldots, Szenario_5$. Für drei Szenarien, auch die in der Abbildung dargestellten Szenarien 0 und 5, wird T_j auf den Wert 500 gesetzt. Die restlichen drei Szenarien haben $T_j = 100$ Zeitschritte zur Verfügung. Insgesamt sind für die Durchführung einer Episode $1800 = 3 \cdot 500 + 3 \cdot 100$ Zeitschritte vorgegeben. Die Anzahl der durchgeführten Episoden liegt bei 1000. Für drei Szenarien wird als bevorzugte Richtung die Richtung im Uhrzeigersinn gewählt, für die restlichen drei die Richtung gegen den Uhrzeigersinn. Für das Training der reaktiven Fähigkeit "Hindernis umfahren" liegt die maximale Anzahl der Individuen in einer Population bei 300.

Zwei der verwendeten sechs Szenarien sind beispielhaft in Abbildung 6.3 dargestellt. Die jeweilige Startrichtung und Startposition des Agenten sind mit einem Dreieck, das den Agenten symbolisiert, gekennzeichnet. In den erstellten Szenarien wird der Agent nah an eine Wand gestellt, siehe Szenario 0, und wird damit von Anfang an mit einer gefährlichen Situation konfrontiert. Wenn dieser Gefahrensituation aus-

gewichen wird, kann sich das Individuum, das durch ein kNN gesteuert wird, nur im freien Raum drehen und einen positiven Fitnesswert ansammeln. Wenn das Individuum mit der aufgebauten Gefahrensituation nicht zurechtkommt und nicht schnell genug reagiert, wird er mit der Wand kollidieren und stehen bleiben. In diesem Fall wird das durch ein kNN repräsentiertes Individuum keinen positiven Fitnesswert sammeln und keine gute Gesamtbewertung erhalten. Ein weiteres Beispiel-Szenario ist in Abbildung 6.3, Szenario 5 abgebildet. Der Agent soll durch einen engen Pass fahren, um einen guten Fitnesswert in diesem Szenario 5 zu erreichen.
Für das Erzielen der reaktiven Fähigkeit ”Hindernis rechts umfahren” und ”Hindernis links umfahren”



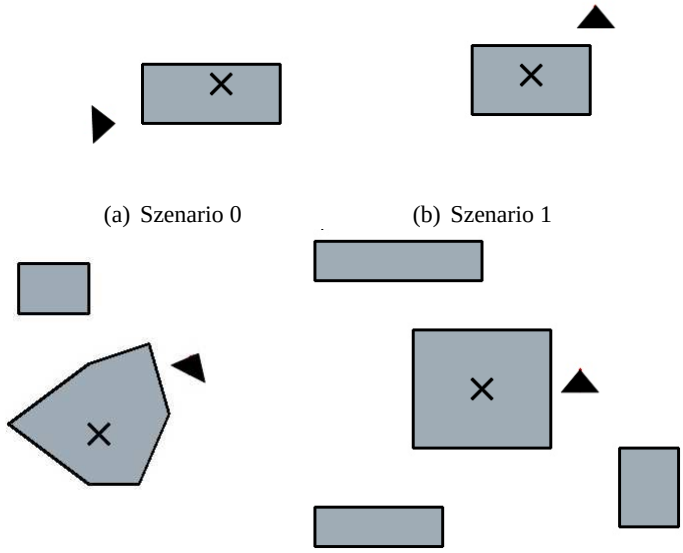
(a) Szenario 0 (b) Szenario 5

Abbildung 6.3: Beispiele für verwendete Szenarien zum Erzielen der reaktiven Fähigkeit ”Hindernis umfahren”. Das Ziel ist es den Roboter so schnell wie möglich vom Hindernis wegzusteuern. Durch das Dreieck wird der Roboter-Agent mit seiner Startrichtung angedeutet.

werden jeweils elf Szenarien erstellt. Die reaktiven Fähigkeiten ”Hindernis rechts/ links umfahren” sollen andere Fertigkeiten als die reaktive Fähigkeit ”Hindernis umfahren” aufweisen. Jedes der elf angewendeten Szenarien erhält $T_j = 1000$ Zeitschritte zur Ausführung. Die Anzahl der durchgeführten Episoden ist 1000. Die maximale Population kann aus 300 Individuen bestehen. Der Agent soll an der tatsächlichen Hindernisform in der vorgegebenen Richtung entlang fahren. In diesem Fall werden für das Training verwendete Szenarien mit einer gewünschten Richtung vorgegeben. Die Winkelfitness wird für das Erzielen der gewünschten, reaktiven Fähigkeiten eine größere Gewichtung erhalten. Diese reaktiven Fähigkeiten werden im DPA-RL-Agenten angewendet.
In Abbildung 6.4 sind die Beispiele der verwendeten Szenarien dargestellt. Die gleichen Szenarien werden für das Erzielen von beiden reaktiven Fähigkeiten verwendet. Für die reaktive Fähigkeit ”Hindernis rechts umfahren” wird die gewünschte Richtung o^{soll} in allen Szenarien auf ”im Uhrzeigersinn” gesetzt. Für das gewünschte reaktive Verhalten ”Hindernis links umfahren” wird die gewünschte Richtung o^{soll} auf den Wert ”gegen den Uhrzeigersinn” gesetzt.

6.2.4 Erzielte kNN für gewünschte reaktive Fähigkeiten

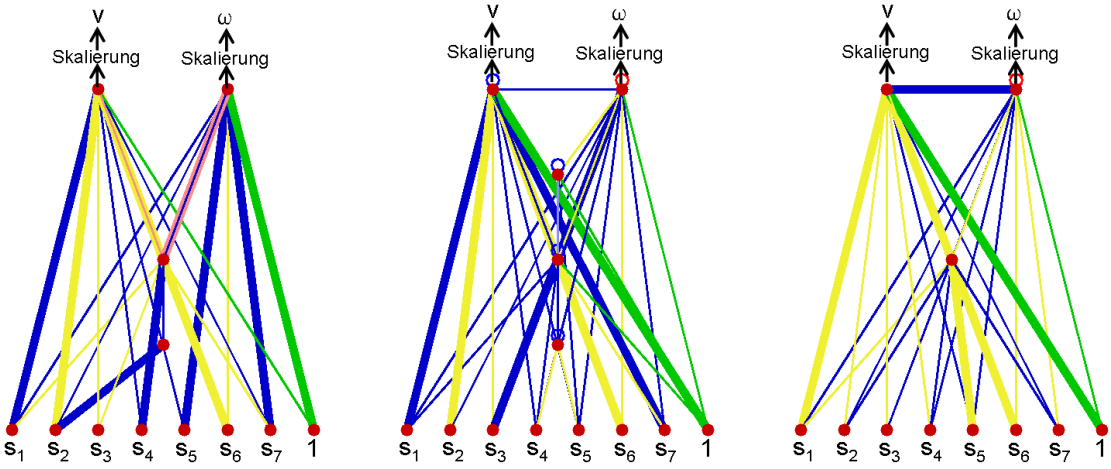
Die Kontrollbefehle werden durch Propagierung der Abstandsmessungen aus Infrarotsensoren erzeugt. Der evolutionäre Prozess wird in der realitätsgetreuen Simulation durchgeführt, so dass eine Art Filterung der eingegebenen Sensordaten schon im trainierten kNN enthalten ist, und die Daten direkt ohne künstlich eingebaute Filter durch das entstehende Netz propagiert werden können. Die Netze zeigen dabei ein gutes Verhalten, vergleiche Kapitel 6.4. Mit dieser Methode wurden kNN mit dem gewünschten Verhalten erzielt und in der Realität angewendet, siehe auch [Köpsel et al., 2007].



(a) Szenario 0 (b) Szenario 1

(c) Szenario 2 (d) Szenario 5

Abbildung 6.4: Beispiele für verwendete Szenarien zum Erzeugen der reaktiven Fähigkeiten ”Hindernis rechts/links umfahren”. Das Ziel beim Erzielen dieser reaktiven Fähigkeiten ist das Fahren entlang des Hindernisses in vorgegebener Richtung. Das Dreieck kennzeichnet den Roboter-Agenten mit seiner Startrichtung. Das Kreuz kennzeichnet die ”Mitte der Welt”. Ausgehend von diesem Punkt wird die Winkelfitness berechnet.



(a) Hindernis umfahren (b) Hindernis rechts umfahren (c) Hindernis links umfahren

Abbildung 6.5: Beispiele für Netzwerke, die eine reaktive Kontrollfunktion repräsentieren.

In Abbildung 6.5 sind die in der vorliegenden Arbeit angewendeten Strukturen der kNN abgebildet, die mithilfe der hier präsentierten Szenarien und Fitnessfunktionen erzeugt wurden. Als Input fließen sieben Sensorwerte und ein Bias-Knoten ein, als Output kommen zwei Befehlssignale, eine Geschwindigkeit und ein Drehwinkel heraus. Gelb markierte Kanten deuten eine negative Gewichtung, blau markierte Kanten eine positive Gewichtung und rot/rosa markierte Kanten in der Abbildung 6.5 eine positive Rückkopplung an. Je breiter die Linie ist, desto größer ist der absolute Wert des Gewichtes für die dargestellte Verbindung. Die grünen Linien sind die Bias-Verbindungen. Als Antwort auf Sensorwerte wird

der Steuerbefehl auf Motoren in cm/s bzw. rad/s geliefert. Das neuronale Netz liefert einen Wert, der im Intervall $[0, 1]$ liegt, so dass der Wert aus dem neuronalen Netz entsprechend der vorgegebenen Einstellungen (maximale Vorwärtsgeschwindigkeit und Drehgeschwindigkeit)[36] skaliert werden muss.
In Abbildung 6.5 kann in allen drei neuronalen Netzen eine besonders starke direkte Verbindung zwischen dem nach vorne ausgerichteten Sensor s_1 und dem Steuerbefehl für die Vorwärtsgeschwindigkeit v beobachtet werden.

6.3 ”Hindernis umfahren” mit Regelungstechnik

6.3.1 Grundidee

Die benötigten, reaktiven Fähigkeiten ”Hindernis umfahren”, ”Hindernis rechts umfahren” und ”Hindernis links umfahren” können alternativ mithilfe der Regelungstechnik und zweier grundlegender Teilschritte realisiert werden. Der prinzipielle Ablauf der reaktiven Fähigkeit ist in Abbildung 6.6 dargestellt. In der Abbildung sind zwei prinzipielle Teilschritte abgebildet und durch zwei Farben gekennzeichnet. Zuerst soll sich der Agent in Bezug auf das Hindernis richtig ausrichten, siehe Teilschritt ”Drehen”, anschließend entlang des Hindernisses fahren, siehe Teilschritt ”Hindernis entlang fahren”. Diese beiden Teilschritte werden noch weiter unterteilt. Der Schritt Drehen wird je nach vorgegebener Richtung in ”Drehen rechts” und ”Drehen links” aufgeteilt. Dieser Teilschritt wird als vorgegebene Konstante realisiert. ”Hindernis entlang fahren” wird in die Teilschritte ”Hindernis rechts entlang fahren” und ”Hindernis links entlang fahren” aufgeteilt und mithilfe der PID-Regler realisiert.
Aus den vier dargestellten Teilschritten werden die für den DPA-RL-Agenten bzw. RL-Agenten benötigten abstrakten Aktionen zusammengestellt. Für die reaktive Fähigkeit ”Hindernis umfahren”, die im RL-Agenten angewendet wird, ist die bevorzugte Richtung zum Umfahren nicht von Bedeutung. Für die reaktive Fähigkeit (abstrakte Aktion) ”Hindernis links umfahren” bzw. ”Hindernis rechts umfahren”, die im DPA-RL-Agenten angewendet wird, wird hingegen nach der Richtung zum Umfahren unterschieden. Die zur Realisierung der reaktiven Fähigkeit ”Hindernis umfahren” angewendete Baumstruktur ist in

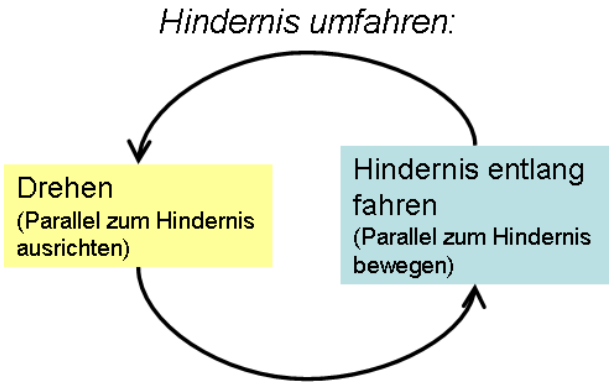


Abbildung 6.6: Prinzipielle Realisierung der Fähigkeit ”Hindernis umfahren” mithilfe einer Schleife

Abbildung 6.7 dargestellt. Alle vier Teilschritte nehmen bei der Bildung dieser reaktiven Fähigkeit teil. Die zwei Grundschritte ”Drehen” und ”Hindernis entlang fahren” sind durch die zwei Farben im Baumknoten gekennzeichnet. In diesem Fall wird in Abhängigkeit von der aktuellen Orientierung des Agenten

[36] maximale Vorwärtsgeschwindigkeit $[0, 1] \propto [0, v_{max}]$, sowie maximale Drehgeschwindigkeit nach rechts bzw. nach links $[0, 1] \propto [-\pi, \pi]$

entschieden, wie es am günstigsten ist sich zu drehen, um die Fahrt entlang des Hindernisses fortzusetzen. Diese Variante wird nur für die beiden in diesem Kapitel untersuchten Beispiel-Szenarien angewendet.
Die für den DPA-RL-Agenten benötigten reaktiven Fähigkeiten ”Hindernis rechts umfahren” und ”Hin-

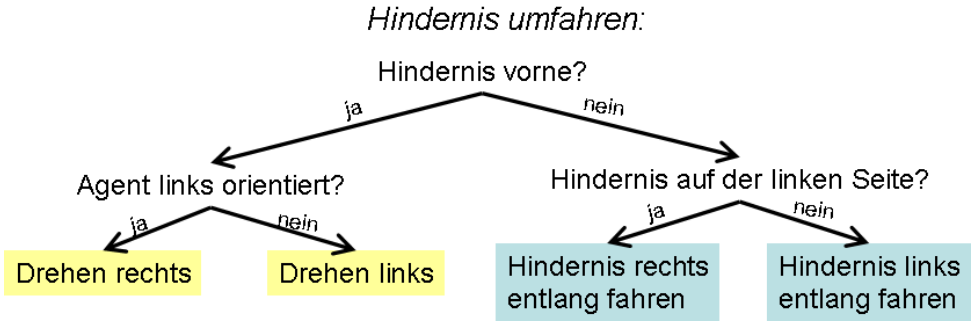


Abbildung 6.7: Realisierung der Fähigkeit ”Hindernis umfahren” mithilfe von vier Regelkreisen in einer fest definierten Struktur des Entscheidungsbaums

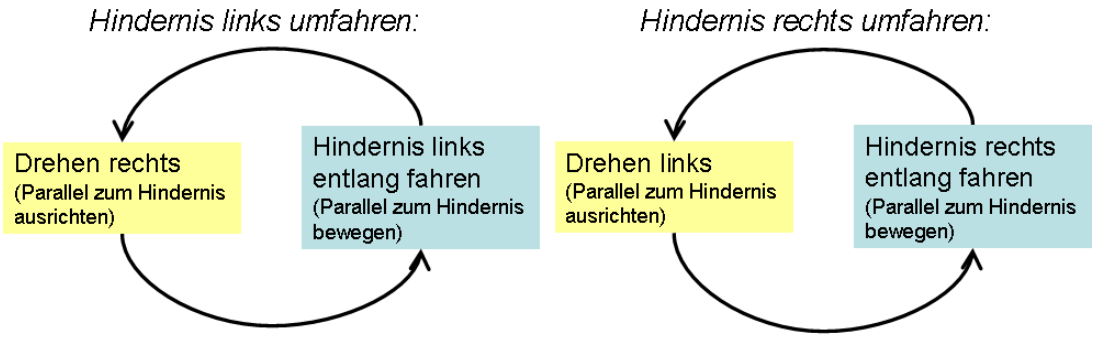


Abbildung 6.8: Prinzipielle Realisierung der reaktiven Fähigkeit ”Hindernis links umfahren” bzw. ”Hindernis rechts umfahren” mithilfe einer Schleife und zwei Regelkreisen

dernis links umfahren” sollen sich in der Ausführungsrichtung unterscheiden. Für beide reaktiven Fähigkeiten werden die gleichen vier Teilschritte wie für die Bildung der reaktiven Fähigkeit ”Hindernis umfahren” verwendet. Die benötigte reaktive Fähigkeit beinhaltet jeweils zwei Teilschritte. Zuerst soll sich der Agent entsprechend der durch die benötigten, reaktiven Fähigkeit vorgegebene Richtung richtig am Hindernis ausrichten, und anschließend soll er sich in vorgegebener Richtung am Hindernis entlang bewegen. Der prinzipielle Ablauf dieser Realisierung ist in Abbildung 6.8 dargestellt, auch hier sind die beiden Grundbestandteile durch die gleichen Farben wie in Abbildung 6.6 gekennzeichnet.

6.3.2 Reaktive Fähigkeit ”Drehe rechts/links”

Der Teilschritt ”Drehe” wird nicht mit Methoden der Regelungstechnik realisiert. Es wird die Tatsache ausgenutzt, dass sich der Agent auf der Stelle drehen kann und der nächste Schritt ”Hindernis entlang fahren” vom Regelmechanismus ausgeführt wird, so dass der Teilschritt ”Drehe” ungenau sein kann. Der durch diesen Teilschritt gesteuerte Agent muss ungefähr seitlich vom Hindernis stehen. Der Teilschritt ”Drehe” gibt eine konstante Drehgeschwindigkeit ω auf die Aktuatoren aus und den Wert Null für die Vorwärtsgeschwindigkeit. Der Teilschritt ”Drehe rechts” gibt den konstanten Wert $(+\omega, 0)$ und der Teilschritt ”Drehe links” den konstanten Wert $(-\omega, 0)$ auf die Aktuatoren aus. Der Teilschritt ”Drehe” wird

so lange ausgeführt, bis der Agent das Hindernis mit den seitlich ausgerichteten Sensoren spürt. Wenn der Agent den Teilschritt ”Drehe links” ausführt, wird er das Hindernis auf der rechten Seite spüren und als nächsten Teilschritt die Aktion ”Hindernis rechts entlang fahren” vorgeben. Im anderen Fall für die Realisierung der benötigten, reaktiven Fähigkeit ”Hindernis links umfahren” wird zuerst der Teilschritt ”Drehe rechts” und anschließend der Teilschritt ”Hindernis links entlang fahren” aktiviert.

6.3.3 Reaktive Fähigkeit ”Hindernis rechts/links entlang fahren”

Der für die reaktive Fähigkeit ”Hindernis umfahren” benötigte Teilschritt ”Hindernis entlang fahren” wird in zwei Teilschritte aufgeteilt: ”Hindernis links entlang fahren” und ”Hindernis rechts entlang fahren”. Die beiden Teilschritte verfolgen ein Ziel und sind symmetrisch. Diese Teilschritte sind dazu bestimmt parallel zum Hindernis mit einer vorgegebenen Distanz d^{soll} zu fahren. Der Teil der Vorwärtsgeschwindigkeit wird durch den konstanten Wert $v(t) = const$ ersetzt. Der Teil der Drehgeschwindigkeit ω wird mit einem PID-Regler realisiert.

$$\omega(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t)dt + K_d \cdot \frac{\partial e(t)}{\partial t} \tag{6.4}$$

Die Regelabweichung $e(t)$ gibt dabei die Differenz zwischen der angestrebten Distanz d^{soll} und der tatsächlichen Distanz zum Hindernis d^{ist} an. Die Soll-Distanz d^{soll} wird in dieser Arbeit auf den Wert $40\ cm$ gesetzt. Wenn sich der Abstand zur Wand verringert, fährt der Agent auf die Wand zu, wenn sich der Abstand vergrößert, fährt der Agent von der Wand weg. Der durch den P-Anteil berechnete Wert ist proportional zur Regelabweichung $e(t)$. Der I-Anteil bezieht die Zeitdauer der Regelabweichung mit ein und führt schließlich zum ausgeglichenen Verhalten. Schließlich wird durch den differentiellen Anteil des Reglers die Änderungsgeschwindigkeit der Regelabweichung mitberücksichtigt.
Die Distanz zum Hindernis auf der rechten Seite wird aus den Sensorwerten von s_2, s_6 und s_4 berechnet. Der Sensor s_4 liegt senkrecht zur Hauptachse des Roboters, die an seiner Fahrtrichtung ausgerichtet ist. Aus diesem Grund liefert der Sensor s_4 den wahren Abstand zur Seite des Roboters. Um die Werte der verwendeten Sensoren besser miteinander abgleichen zu können, werden Werte aus den Sensoren s_2 und s_6 auf den Sensor s_4 projiziert. Die projizierten Werte werden als d'_{s_2} und d'_{s_6} bezeichnet. Der im Regler angewendete Wert d^{ist} wird dann wie folgt berechnet: $d^{ist} := \min\{d'_{s_2} d'_{s_6}, d_{s_4}\}$. Wenn einer der Sensoren eine maximale Distanz liefert, also $d_{s_{\{2,6\}}} = d^{max}$, dann wird die Projektion auch als maximaler Wert angerechnet: $d'_{s_{\{2,6\}}} = d^{max}$. In Abbildung 6.9 wird die Berechnung des Wertes d^{ist} beispielhaft veranschaulicht. Der projizierte Wert aus dem Sensor s_2 liefert die maximal mögliche Distanz d_{max}, da mit diesem Sensor kein Hindernis gemessen wird. Der Wert aus dem Sensor s_6 wird in Abbildung 6.9 mit d'_{s_6} bezeichnet.
Die Distanz d^{ist} zum Hindernis auf der linken Seite wird aus den Sensoren s_3, s_5 und s_7 analog zur rechten Seite berechnet.
Dieser Teilschritt der Regelung gibt die folgende Aktion aus $(\omega, v(t))$. Die Schwierigkeit besteht dabei in der richtigen Bestimmung der Kalibrierparameter K_p, K_I, K_d. Dieser Schritt wird in der realitätsnahen Simulation manuell durchgeführt.
Zuverlässige Messwerte der IR-Sensoren sind eine der Voraussetzungen zur Realisierung der reaktiven Fähigkeiten mithilfe der Regelungstechnik. Der I-Anteil kann das Rauschen nur begrenzt unterdrücken. Für die in der vorliegenden Arbeit angewendeten IR-Sensoren wird eine zusätzliche Filterung der gemessenen Werte benötigt. Diese Tatsache ist durch zahlreiche Experimente zum Vorschein gekommen. Die Messwerte werden dabei in vier Messzyklen zwischengespeichert, Ausreißer werden eliminiert und schließlich wird der arithmetische Mittelwert gebildet.

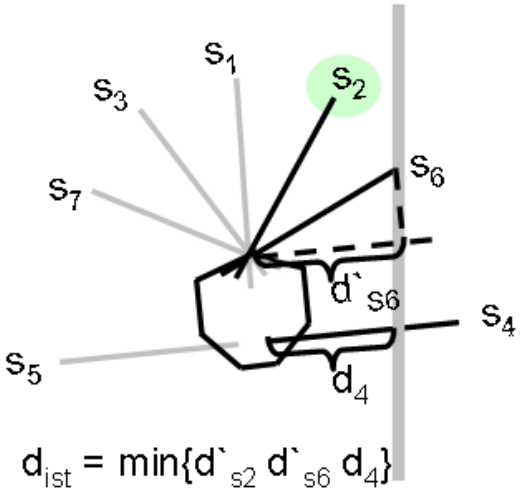


Abbildung 6.9: Verwendete Größen zur Realisierung der Fähigkeit ”Hindernis entlang fahren”

6.4 Realisierung von ”Hindernis umfahren” mit NEAT und einem PID-Regler

Die zwei oben beschriebenen Realisierungen der reaktiven Fähigkeit ”Hindernis umfahren” werden miteinander verglichen. Für den gewünschten Vergleich werden zwei einfache Beispiel-Szenarien aufgestellt und ein manuell erstellter Entscheidungsbaum angewendet. Die erzielten Ergebnisse werden zur Entwicklung des DPA-RL-Agenten beitragen. Die entwickelte Struktur war für komplexere Szenarien nicht ausreichend, so dass als Lösung die Zerlegung der abstrakten Aktion ”Hindernis umfahren” in zwei feinere abstrakte Aktionen ”Hindernis rechts/links umfahren” durchgeführt wurde. Der fest programmierte Entscheidungsbaum hat nicht die benötigte Flexibilität gezeigt. Die Notwendigkeit des Lernschrittes für die situationsabhängige Betätigung der abstrakten Aktionen ist deutlich zum Vorschein gekommen.

6.4.1 Entscheidungsbaum

Die angewendeten Beispiel-Szenarien sind einfache Navigationsprobleme mit einem sich im Sichtfeld des Agenten befindenden Ziel. Die Planung erfolgt durch Propagierung der aus der Kamera und IR-Sensoren gewonnenen Informationen über den vorgegebenen Entscheidungsbaum, vergleiche Abbildung 6.10. Mithilfe des Entscheidungsbaums wird eine Entscheidung in Abhängigkeit von aktuellen Informationen für eine der vorhandenen, alternativen Aktionen getroffen. Das Ziel ist durch eine vorgegebene Farbe spezifiziert. Die abstrakten Aktionen, also die in Blätter des Baumes verankerten Alternativen, besitzen die ermittelten, reaktiven Fähigkeiten.
Der entwickelte Entscheidungsbaum beinhaltet vier Arten von Blättern. Die erste Art der Blätter ist eine der beiden Realisierungen der abstrakten Aktion ”Hindernis umfahren” [37]. Die zweite Art der Blätter ist die abstrakte Aktion ”zum Ziel fahren”, deren Realisierung auf Bildinformationen basiert, siehe auch Abschnitt 6.1.2. Die dritte Aktion ist ”Ziel suchen”, die den Agenten mit vorgegebener, konstanter Geschwindigkeit um $360°$ dreht, um das aus dem Bild verlorene Ziel wiederzufinden. Die letzte Aktion ”Anhalten” hält den Agenten an.
Der Ablauf der Entscheidungsfindung wird im Folgenden kurz beschrieben. Zunächst wird aus dem Ka-

[37] aus vier Teilschritten zusammengesetzter PID-Regler, im Folgenden PID-Regler genannt, oder erzieltes künstliches neuronales Netz

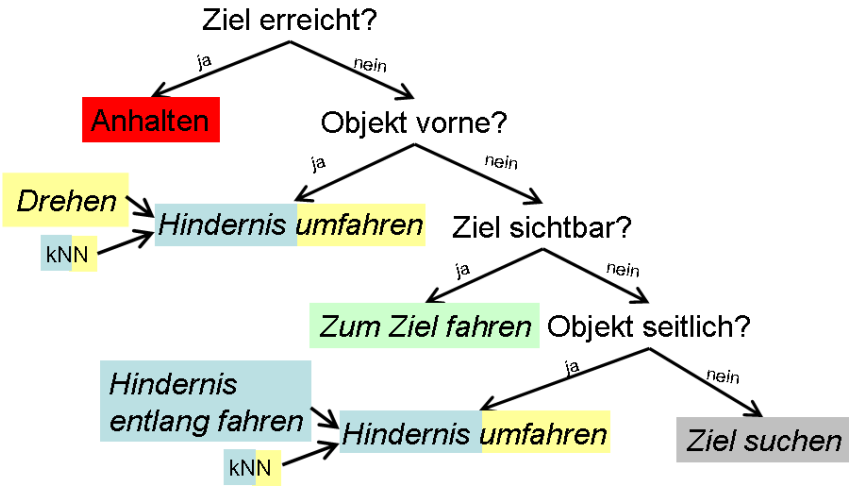


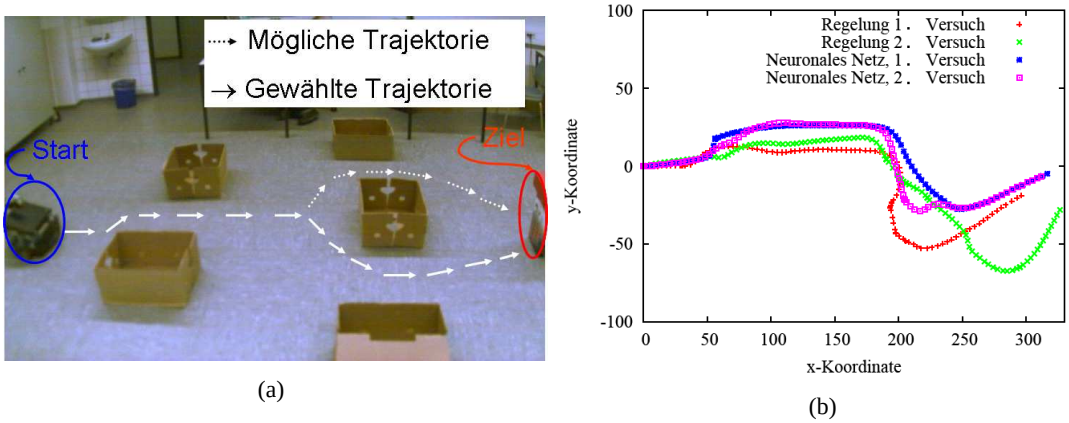
Abbildung 6.10: Verwendeter Entscheidungsbaum zur Lösung einfacher Navigationsprobleme. Die Aktion ”Hindernis umfahren” kann mit Regelungstechnik oder kNN realisiert werden. Die Realisierung mit Regelungstechnik besteht aus zwei Teilschritten: ”Drehen”, durch ein gelbes Kästchen gekennzeichnet, und ”Hindernis entlang fahren”, durch ein blaues Kästchen gekennzeichnet. Die Realisierung mit kNN besteht aus nur einer Funktion, die beide Teilschritte direkt beinhaltet und durch ein zweifarbiges Kästchen gekennzeichnet ist. Die Pfeile von beiden möglichen Realisierungen bedeuten, dass an dieser Stelle eine der Möglichkeiten gewählt werden kann.

merabild die Information über das Ziel extrahiert. Dann wird entschieden, ob das Ziel erreicht ist. Wenn dieser Fall eintritt, hält der Roboter an, Aktion ”Anhalten”. Wenn der Agent das Ziel nicht erreicht hat, wird geprüft, ob ein Hindernis in unmittelbarer Nähe des Agenten liegt und die Weiterfahrt des Agenten in Richtung des Ziels verhindert. Die Prüfung erfolgt mithilfe der nach vorne ausgerichteten IR-Sensoren des Roboters s_1, s_2, s_3. Wenn dieser Fall auftritt, wird die abstrakte Aktion ”Hindernis umfahren” aktiviert. Je nach Realisierung wird entweder ein kNN die Steuerung des Agenten für den vorgegebenen Zeitabschnitt übernehmen und den Agenten entsprechend drehen, oder der Teilschritt ”Drehen” des PID-Reglers wird die Kontrolle über den Agenten übernehmen. Wenn kein Objekt vor dem Roboter-Agenten liegt und das Ziel sichtbar ist, wird die Aktion ”zum Ziel fahren” eingesetzt. Im anderen Fall, wenn sowohl ein Hindernis im Weg ist als auch das Ziel nach der neuen Ausrichtung des Roboters aus dem Blickfeld der Kamera verschwunden ist, wird geprüft, ob ein Objekt seitlich vom Agenten liegt, siehe Teilschritt ”Objekt seitlich?”. Wenn das der Fall ist, wird die Aktion ”Hindernis umfahren” fortgesetzt. Entweder wird wieder das kNN eingesetzt oder der zweite Schritt des PID-Reglers ”Hindernis entlang fahren” wird fortgesetzt. Im anderen Fall wird die Aktion ”Ziel suchen” ausgeführt. Die Aktion ”Hindernis umfahren” bekommt einen vorgegebenen Zeitabschnitt (Beständigkeitszeit) für die Ausführung der abstrakten Aktion ”Hindernis umfahren”.

6.4.2 Ergebnisse

Eine der beiden möglichen Realisierungen wird im oben dargestellten Entscheidungsbaum an der Stelle ”Hindernis umfahren” so substituiert, dass zwei Entscheidungsbäume entstehen, die sich in zwei Blättern unterscheiden. Jeder Versuch in einem Szenario mit einem der zwei Entscheidungsbäume wurde 10 Mal durchgeführt. Insgesamt sind 40 Versuche in der Realität gestartet worden. Die in einem Szenario erziel-

ten Trajektorien ähneln einander. Das Ziel wurde mit beiden Entscheidungsbäumen in jedem Szenario in ungefähr gleicher Zeit erreicht.
Das erste Beispiel-Szenario 1 ist in Abbildung 6.11 (a) dargestellt. Für dieses Szenario wird durch den Einsatz des Entscheidungsbaums, mit Realisierung der abstrakten Aktion ”Hindernis umfahren” durch ein kNN, ein gleichmäßigerer Verlauf der Trajektorie vermutet. Die gefahrenen Trajektorien sind etwas glatter als die mit der Realisierung mit Regelungstechnik (PID-Regler) erzielten Trajektorien, siehe Abbildung 6.11 (b).
Das zweite angewendete Beispiel-Szenario 2 ist in Abbildung 6.12 (a) dargestellt. In diesem Szenario



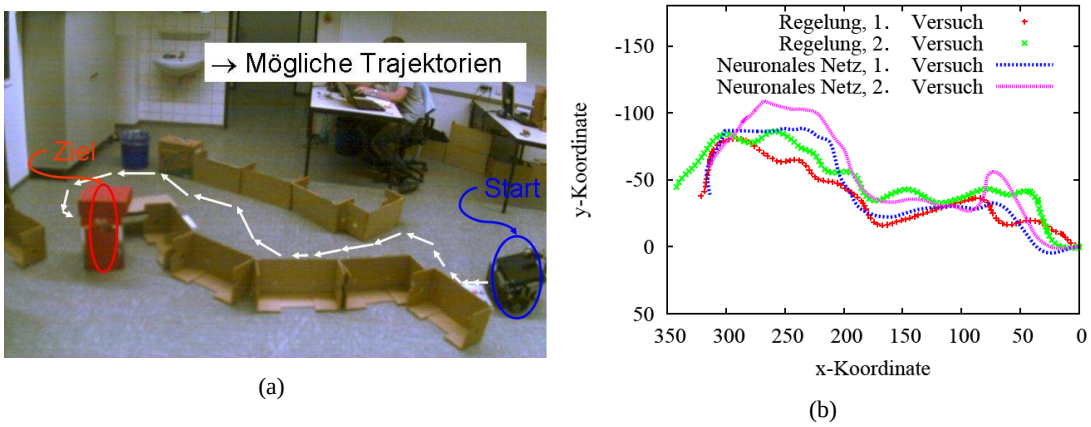
(a) (b)

Abbildung 6.11: Bild (a): Beispiel-Szenario 1 in der Realität. Bild (b): vom Roboter gefahrene beispielhafte Trajektorien

kann eine bessere Flexibilität der mit einem kNN realisierten abstrakten Aktion ”Hindernis umfahren” beobachtet werden. Für eine Realisierung der abstrakten Aktion ”Hindernis umfahren” ist der aufgebaute Tunnel zu eng gewesen. Um den Versuch fortsetzen zu können, wurde der aufgebaute Tunnel erweitert. Der Verlauf der erzielten Trajektorien bei Einsatz des kNNs zur Realisierung der reaktiven Fähigkeit ”Hindernis umfahren” ist mehr gekrümmt, vergleiche die vier Trajektorien in Abbildung 6.12 (b). Dieses Ergebnis lässt sich durch eine leichte Veränderung der Umgebung erklären, da zuerst die Versuchsreihen für die Realisierung mit einem kNN und anschließend die Versuchsreihen für die Realisierung mit einem PID-Regler durchgeführt worden sind.
Um den Unterschied der erzielten Trajektorien zu verifizieren, wird das Messkriterium der intrinsischen Energie des Trajektorienverlaufs aufgestellt. Dieses Messkriterium drückt aus, wie glatt der Trajektorienverlauf ist, siehe Gleichung (6.5).

$$E_{intr}(x,y) = \int_0^T \sqrt{\left(\frac{\partial^2(x(t),y)}{\partial t \partial t}\right)^2 + \left(\frac{\partial^2(x,y(t))}{\partial t \partial t}\right)^2}\, dt \tag{6.5}$$

wobei (x, y) der Trajektorienverlauf ist. In der vorliegenden Arbeit wird das Integral durch eine Integralsumme approximiert, T ist die Anzahl der Zeitschritte bis zum Erreichen des Ziels. Die intrinsische Energie wird insgesamt über zehn gefahrene Trajektorien für die jeweilige Methode für das Beispiel-Szenario 1 in Abbildung 6.11 und das Beispiel-Szenario 2 in Abbildung 6.12 gemittelt.
Die intrinsischen Energien (absolute und relative Werte) für die jeweilige Methode sind in Tabelle 6.1 zusammengestellt. Der mittlere Wert der Glattheit des Trajektorienverlaufs bei Einsatz des kNNs ist im Vergleich zum Einsatz eines PID-Reglers um -4% beim Messkriterium intrinsische Energie schlechter, siehe erste Zeile in Tabelle 6.1. Die Ergebnisse für das Beispiel-Szenario 2 weisen einen noch kleineren



(a) (b)

Abbildung 6.12: Bild (a): Beispiel-Szenario 2 in der Realität. Bild (b): vom Roboter gefahrene beispielhafte Trajektorien

Tabelle 6.1: Vergleich der gefahrenen Trajektorien für unterschiedliche Realisierungen der reaktiven Fähigkeit ”Hindernis umfahren”, zwei Szenarien absolute und relative Werte.

Szenario	$E_{intr,}$ kNN	$E_{intr,}$ PID-Regler	E_{intr} in %	t-Test
Beispiel-Szenario 1	80.30	76.56	-4.66%	0
Beispiel-Szenario 2	72.6	73.67	1.45%	0

Unterschied auf, vergleiche Zeile 2 in Tabelle 6.1. In diesem Fall liegt eine Verbesserung um 1.45% für den Trajektorienverlauf, der mit einem kNN erzielt worden ist, vor. Die erzielten Unterschiede für zwei mögliche Realisierungen sind für beide Szenarien nicht signifikant, da die Unterschiede sehr klein sind, und die zur Bestimmung der Ergebnisse verwendete Statistik für die jeweilige Konfiguration nur aus zehn Versuchen besteht. Beide vorgeschlagenen Realisierungen können angewendet werden.
Die Machbarkeit einer automatischen Generierung der reaktiven Fähigkeit ”Hindernis umfahren” mithilfe von evolutionären Algorithmen konnte damit gezeigt werden. Die Anzahl der bis zum Ziel benötigten Schritte für beide Konzepte bleibt ungefähr gleich.
Die Bestimmung der durch ein kNN repräsentierten, reaktiven Fähigkeit ”Hindernis umfahren” mithilfe des evolutionären Ansatzes ist für den Anwender viel intuitiver, erfordert keine Programmierung, benötigt keine Aufbereitung der Sensorwerte und keine zeitaufwendigen Kalibrierschritte und kann sofort nach dem Simulationsschritt angewendet werden. Es genügt die Fitnessfunktion aufzustellen und die passenden Szenarien für die Simulation zu konstruieren, um die gewünschten Fähigkeiten zu erzielen. Einer der Nachteile des vorgeschlagenen Verfahrens liegt in der für die Simulation benötigten Zeit, wobei dieser Schritt auf einem anderen Rechner parallel ausgeführt werden kann und keine Anwesenheit des Nutzers benötigt, was hingegen bei einer Kalibrierung der Fall ist. In vielen Fällen ist es einfacher, bestimmte Beispiele aufzustellen, anstatt das ganze Konzept im Voraus zu durchdenken. Dieser allgemeine Vorteil der evolutionären Algorithmen wurde auch hier ausgenutzt.
Die Einbindung der mit einem PID-Regler repräsentierten abstrakten Aktion ”Hindernis umfahren” in den Entscheidungsbaum weist kein schlechteres Verhalten als die Realisierung durch kNN auf. Die Regelungstechnik ist für viele Nutzer ein vertrauteres Werkzeug als die NEAT-Methode. Eine Veränderung der Roboterkonfiguration im Fall der Anwendung der evolutionären Algorithmen ist weniger schmerzhaft als bei der Regelungstechnik, da die angewendeten Szenarien und Fitnessfunktionen gleich bleiben. Für die Anwendung der Regelungstechnik muss das ganze Konzept vom Anwender überarbeitet werden.

Kapitel 7

Lernen im Navigationsproblem auf höherer Abstraktionsebene

Die möglichen Aktionen, die der rationale Agent zur Auswahl hat, können von unterschiedlichem Abstraktionsgrad sein. Die Aufgabenstellung bleibt unverändert, der Agent soll eine Kette aus Aktionen erlernen, um das vorgegebene Ziel zu erreichen. Im vorliegenden Kapitel wird hauptsächlich der Begriff abstrakte Aktion angewendet, da dieser Begriff in Verbindung mit dem Reinforcement-Learning-Ansatz gebraucht wird. Der Begriff abstrakte Aktion ist mit dem Begriff der reaktiven Fähigkeit stark verwandt. Die abstrakten Aktionen werden durch reaktive Fähigkeiten, die lokale bzw. globale Informationen automatisch in Kontrollbefehle der Motorsteuerung verarbeiten, realisiert. Die Möglichkeit zum Erzielen der benötigten, reaktiven Fähigkeiten wurde in Kapitel 6 beschrieben.
Das Markow-Entscheidungsproblem beinhaltet Aktionen, die alle die gleiche Zeit zur Ausführung benötigen. Bei der Anwendung der abstrakten Aktionen wird diese Bedingung verletzt, so dass neue Randbedingungen entstehen, und das Semi-Markow-Entscheidungsproblem zur Geltung kommt, siehe Kapitel 3.5.
Während der Ausführung der Mission soll der entwickelte Agent die Fähigkeit besitzen, die fließenden Informationen zu verarbeiten und Entscheidungen auf Grundlage des neuen Informationsstands zu treffen, also die Fähigkeit zum Lernen im Onlinebetrieb besitzen. Der entwickelte Agent soll adaptionsfähig sein, die Fähigkeit zur Erkundung der Umgebung besitzen und das Umgebungsmodell erfassen können. Der Agent soll dabei selbstständig durch situationsabhängiges Aktivieren der vorhandenen, abstrakten Aktionen lernen. Der im vorliegenden Kapitel eingeführte Ansatz für das Lernen auf Grundlage von abstrakten Aktionen stellt die endgültige Lösung für das in dieser Arbeit als Benchmark betrachtete Navigationsproblem dar.
In Kapitel 7.1 wird die grundlegende Idee dieses Abschnittes dargelegt. Diese Idee wird durch ein Beispiel mit dem Menschen verdeutlicht. Die Doppelpass-Architektur wird in ihrer ursprünglichen Form mit den wichtigsten Bestandteilen wie Planungsprozess, Ausführungsprozess und Optionenbaum eingeführt. Anschließend wird ein kurzer Überblick über die Arbeiten, die in eine ähnliche Richtung zielen, gegeben.
In Kapitel 7.2 wird der DPA-RL-Agent eingeführt und mit den anderen Agentenarten verglichen. Der konkrete Optionenbaum wird in diesem Kapitel eingeführt und in folgenden Kapiteln verwendet. Die Knotenarten und die beiden wichtigsten Kontrollprozesse werden in diesem Kapitel auch genauer untersucht. Der benötigte Synchronisationsschritt zwischen dem Planungs- und Ausführungsprozess wird eingeführt.
Das Kapitel 7.3 ist der Lernfähigkeit, die eine charakteristische Eigenschaft des DPA-RL-Agenten ist, gewidmet. Der Lernprozess ist vom Aufbau des Aktions- und Zustandsraumes abhängig. Dieses Kapitel beinhaltet die Realisierung der abstrakten Aktionen und den Aufbau des Zustandsraumes.
In Kapitel 7.4 wird der Aufbau der Aktions- und Zustandsräume präsentiert.

7.1 Lernen auf höherer Abstraktionsebene

7.1.1 Einführung

Eine Reihe von Schwierigkeiten ist mit der Realisierung der Idee des Lernens auf Grundlage von abstrakten Aktionen verbunden. Die erste Schwierigkeit liegt in der Realisierung der für die abstrakte Aktion charakteristischen Autonomie. Eine weitere Schwierigkeit, die mit der Anwendung der abstrakten Aktionen verbunden ist, sind eine unvorhersehbare Dauer der abstrakten Aktion und komplexere Bedingungen für die Ausführung solcher abstrakten Aktionen. Die Anwendung der abstrakten Aktionen impliziert den Aufbau eines speziellen Zustandsraums. Für den Lernprozess wird die angewendete Architektur erweitert, die Mechanismen zum dynamischen Aufbau des Zustands- und Aktionsraumes werden eingeführt.
Die Verwaltung und die Bildung der abstrakten Aktionen wird in der vorliegenden Arbeit auf Grundlage der Doppelpass-Architektur (DPA) [Burkhard et al., 2002][38] realisiert. Die DPA gehört zur Familie der hybriden Architekturen. Diese Architektur beinhaltet eine globale Datenstruktur (Optionenbaum) und zwei parallel laufende, ausführende und planende Prozesse. Die DPA stellt nur ein Hilfsmittel dar, um die abstrakten Aktionen und deren Verwaltung und Ausführung zu bilden. Sie beinhaltet in ihrer ursprünglichen Form nicht das für die vorliegende Arbeit wesentliche Merkmal, die Fähigkeit zum Lernen. Aus diesem Grund unterscheidet sich die hier dargestellte Doppelpass-Reinforcement-Learning-Architektur (DPA-RL-Architektur) grundsätzlich von der DPA.
Der Agent, der diese erweiterte DPA-RL-Architektur beinhaltet, wird als DPA-RL-Agent bezeichnet. Der DPA-RL-Agent wird auch in der Diplomarbeit von Andreas Mai [Mai, 2010] eingeführt. In dieser Diplomarbeit werden detaillierte Implementierungsaspekte und ein ausführlicher Vergleich mit der DPA geschildert. In der vorliegenden Arbeit wird mehr auf die Grundidee und auf das Lernen des DPA-RL-Agenten eingegangen. Am Anfang des Lernvorgangs existieren insgesamt drei unterschiedliche abstrakte Aktionen (drei Möglichkeiten, wie der Agent handeln kann) ”zum Ziel fahren”, ”Hindernis rechts umfahren” und ”Hindernis links umfahren”. Mit der Fortsetzung des Lernprozesses kann der Aktionsraum um weitere abstrakte Aktionen erweitert werden. Der Lernprozess des DPA-RL-Agenten wird auf Grundlage der präsentierten Aktionsmenge realisiert.
Das in der vorliegenden Arbeit behandelte Navigationsproblem setzt das Fehlen des Wissens über den Aufbau der Umgebung, in der der Agent agiert, voraus. Im entwickelten DPA-RL-Agenten wird angenommen, dass die Position des Ziels im Vorfeld bekannt ist. Der generische Optionenbaum, der die abstrakten Aktionen und Mechanismen zur Steuerung dieser Aktionen beinhaltet, ist auch vorgegeben. Gelernt werden die Entscheidungen, welche abstrakten Aktionen sich in welchen Situationen als beste Möglichkeit herausgestellt haben. Die möglichen abstrakten Aktionen werden über einen Knoten im Optionenbaum verbunden. Dieser Knoten liegt oberhalb der Ebene im Optionenbaum, wo abstrakte Aktionen angesiedelt sind. Die Prioritätenfunktion dieser Knoten wird mit einer RL-Methode erlernt.
Die DPA-RL-Architektur ist eine Realisierung der flexiblen und anpassungsfähigen, hybriden Architektur für den lernbasierten, rationalen Agenten. Diese Architektur wird für die Lösung des Navigationsproblems entwickelt, kann aber ohne Schwierigkeiten für andere Probleme angewendet werden. Mit dem DPA-RL-Agenten werden sehr gute Ergebnisse für das Navigationsproblem in realen Umgebungen erzielt. Der entwickelte Ansatz ermöglicht im Vergleich zum Ansatz aus dem vorherigen Kapitel 5 ein viel schnelleres Lernen in deutlich komplexeren Umgebungen.

[38] Diese Architektur wird aus zwei Gründen als Doppelpass-Architektur bezeichnet. Sie wurde ursprünglich für Fußball-Roboter entwickelt. Die Fähigkeit einen Doppelpass auszuführen ist für Fußball-Roboter für den Erfolg von entscheidender Bedeutung. Der zweite Grund liegt darin, dass zwei parallel laufende Prozesse in der vorgegebenen Datenstruktur laufen. Diese Prozesse können auch als Pässe genannt werden, siehe auch [Berger, 2006, Kap. 3, S. 31].

7.1.2 Beispiel

Zur besseren Nachvollziehbarkeit wird die verfolgte Idee anhand eines Beispiels über das Verhalten eines Menschen im Straßenverkehr veranschaulicht. Das Vorgehen eines Menschen und des hier dargestellten Agenten sind ähnlich.
Die Navigation in den Straßen besteht zwar aus ganz vielen Teilschritten. Diese Teilschritte können aber zu wichtigen abstrakten Schritten gruppiert werden. Zum Beispiel müssen mehrere Regelkreise im Körper eines Menschen ablaufen um die Aktion ”Straße entlang laufen” auszuführen. Zuerst soll das rechte, dann das linke Bein nach vorne gesetzt werden. Das Gleichgewicht soll ständig geprüft werden. Wenn nötig, sollen Korrekturen vorgenommen werden, um den Mindestabstand zur Wand oder zum Straßenrand nicht zu unterschreiten. Die Aktion ”Straße entlang laufen” besteht zwar aus mehreren Teilschritten, diese haben aber keinen Einfluss auf die Planung des Weges zum Ziel.
Ein Mensch läuft solange entlang einer Straße, die keine Verzweigungen aufweist, bis die nächste Quer-

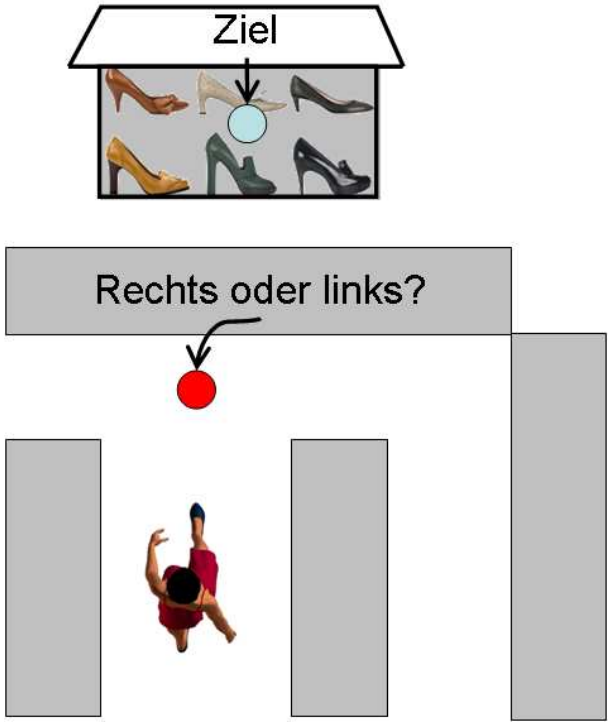


Abbildung 7.1: Der Mensch wird erst dann veranlasst eine Entscheidung zu treffen, wenn mehrere mögliche Aktionen zur Verfügung stehen. Der rote Punkt markiert den Entscheidungsknoten.

straße kommt, also Alternativen zur Verfügung stehen, siehe roten Entscheidungspunkt in Abbildung 7.1. Erst an einer Verzweigung wird über die nächste mögliche Alternative entschieden. Die möglichen Alternativen melden sich selbst, wenn sie ausführbar sind. Im abgebildeten Beispiel sind die Alternativen am Entscheidungspunkt ”nach rechts abbiegen” oder ”nach links abbiegen”. Wenn die falsche Entscheidung im Entscheidungsknoten getroffen wird, im Beispiel statt nach links, nach rechts abgebogen wird, führt diese Entscheidung zum Misserfolg. Der Mensch wird einen Kreis um den Block drehen. Dann wird wieder der Entscheidungspunkt erreicht. Beim nächsten Mal wird die neue Alternative ausprobiert. Diese Alternative ist die richtige Entscheidung ”nach links abzubiegen” und führt den Menschen zum Ziel. Es wird eine neue Alternative ausprobiert, da die andere Alternative zum Misserfolg führte.
Die Entscheidungen über die neuen ausführbaren Aktionen (Alternativen) werden in dem Fall getroffen,

wenn die gewählte Aktion nicht weiter ausführbar ist, oder die Zeit (Geduld) für die Ausführung dieser Aktion überschritten ist. Die Entscheidung über die nächstbeste ausführbare Aktion wird in Abhängigkeit von den gesammelten Erfahrungen, falls welche vorhanden sind, getroffen.
Der aufgebaute Agent hat auch zuerst drei mögliche Aktionen ”zum Ziel fahren”, ”Hindernis rechts entlang fahren” und ”Hindernis links entlang fahren”. Der Agent wird sich auch zuerst, wenn mehrere Alternativen zur Verfügung stehen, zufällig für eine der Alternativen entscheiden. Der Agenten wird aber ein Belohnungssignal empfangen, wenn die Aktion ausgeführt wird. Auf dieser Grundlage wird die Nutzenfunktion mit dem SARSA-Algorithmus sequentiell adaptiert. Die Nutzenfunktion repräsentiert die Prioritätenfunktion, die mögliche Alternativen nach ihrer Nützlichkeit sortiert. Der Agent speichert die gesammelten Erfahrungen in der Nutzenfunktion und zeigt beim nächsten Durchlauf ein besseres Verhalten.

7.1.3 Doppelpass-Architektur

Die DPA wird als Grundlage für die entwickelte Architektur gewählt, weil sie eine gute Erweiterbarkeit bietet und hierarchisch aufgebaut ist, was gut zur Bildung der abstrakten Aktionen und deren Verwaltung passt. Gute Ergebnisse mit der Anwendung der DPA wurden im RoboCup erzielt.[39] Die Idee der Doppelpass-Architektur (DPA) wurde in [Burkhard et al., 2002] vorgestellt. Eine detailliertere Beschreibung mit Implementierungsaspekten findet sich in der Diplomarbeit von Berger, siehe [Berger, 2006]. Die DPA repräsentiert ein Leistungselement im DPA-RL-Agenten, vergleiche Abbildung 7.4.
Die DPA repräsentiert eine Art BDI-Agent mit ein paar Besonderheiten. Die DPA verwendet Konzepte der BDI-Modelle als Basis für die Datenstruktur und die über die Datenstruktur laufenden Prozesse. Die Prozesse der Vorbereitung und Verfeinerung der verfolgten Pläne werden durch die Zustände der Umgebung, die vorhandenen Informationen (Beliefs), die durch die verfolgten Ziele geprägten Wünsche (Desires) und den eigentlichen Plan (Intention) beschrieben.
Langfristige Planung auf der einen und Reaktivität und Echtzeitfähigkeit auf der anderen Seite sind generelle Eigenschaften der DPA. Die Planung und Reaktivität besitzen in der DPA eine eigene Laufzeitorganisation und sind über die Datenstruktur in Form des Optionenbaums miteinander verbunden, siehe Abbildung 7.2.
Wichtige Merkmale der DPA sind:

- Hierarchische Organisation der zentralen Datenstruktur. Langfristige Pläne können aus kurzfristigen Plänen zusammengestellt werden. Diese Datenstruktur wird in Form eines Optionenbaums realisiert.
- Die Entscheidungsfindung und Entscheidungsausführung geschieht in zwei unabhängigen Prozessen, dem Ausführungsprozess und Planungsprozess.
- Die beiden Prozesse sind zwei unabhängige Prozesse, die den gesamten Optionenbaum von der Wurzel bis zu den Blättern des Baumes traversieren (Top-Down-Prozesse), siehe Abbildung 7.2.
- Die DPA orientiert sich an der nebenläufigen Planer-Reaktor-Architektur, siehe Abbildung 7.3

Die zentrale Datenstruktur der DPA und der DPA-RL-Architektur ist ein Optionenbaum. Der Optionenbaum wird sowohl zur Planung als auch zur Ausführung der erstellten Pläne verwendet. In der DPA

[39] Die DPA hat gute Ergebnisse in der RoboCup-Simulationsliga erzielt, u.a. den 6. Platz bei der Weltmeisterschaft in Atlanta 2007 [Redlich & Henze, 2007].

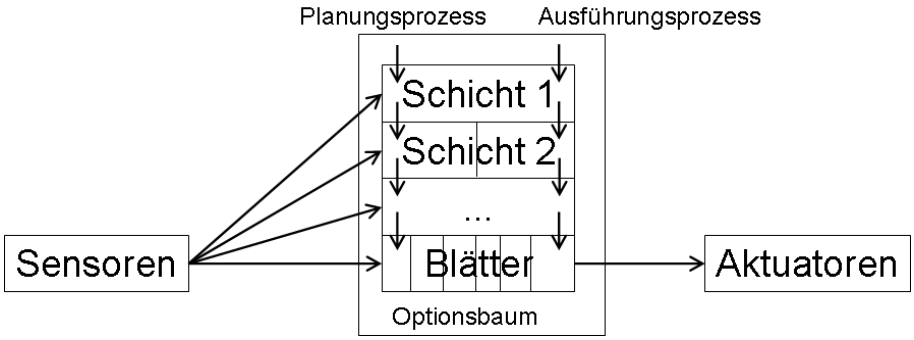


Abbildung 7.2: DPA bestehend aus einer zentralen Datenstruktur (Optionenbaum) und zwei parallel laufenden Ausführungs- und Planungsprozessen, nach Berger [Berger, 2006, Kap. 3.2.3]

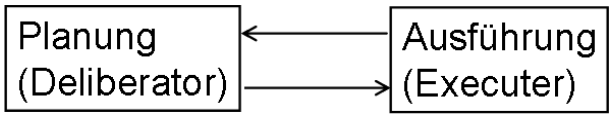


Abbildung 7.3: Nebenläufige Planer-Reaktor-Architektur, nach Arkin [Arkin, 1998, Kap. 6.4]

existieren drei Arten der Knoten aus denen der Optionenbaum zusammengestellt wird. Diese Arten sind Sequenz-Knoten, Choice-Knoten (Auswahl-Knoten) und Blätter (Basis-Knoten). In der DPA-RL-Architektur wird zusätzlich der RL-Choice-Knoten eingeführt. Der RL-Choice-Knoten als vierte Knotenart ermöglicht die gewünschte Lernfähigkeit.
Die Trennung der hierarchisch aufgebauten Datenstruktur (Optionenbaum) und der daran geknüpften Kontrollprozesse (Planungs- und Ausführungsprozesse) löst viele Kontext- und Echtzeitprobleme. Die Dynamik des Systems wird durch Anwendung dieser beiden Kontrollprozesse realisiert.
Der Planungsprozess bereitet die Pläne für den Ausführungsprozess vor, der diese Pläne ausführt. Da die erzielten Pläne in der Regel eine gewisse Beständigkeit haben und damit eine Lebenszeit von deutlich mehr als einem Zeitschritt aufweisen, kann der Ausführungsprozess mehrere Zeitschritte lang unabhängig vom Planungsprozess arbeiten. Um dabei eine kontinuierliche Entscheidung über die auszuführenden Aktionen zu ermöglichen, kann der Ausführungsprozess den Planungsprozess in festen Intervallen aufrufen. Der Ausführungsprozess prüft auch die Möglichkeit, die von der Planung ausgewählte Handlung auszuführen. Wenn diese nicht ausführbar ist, ruft der Ausführungsprozess den Planungsprozess auf, um einen neuen Plan, der an die aktuelle Situation angepasst ist, zu erstellen.

7.1.4 Grundidee

Mithilfe der DPA-RL-Architektur werden die abstrakten Aktionen verwaltet und der Lernprozess auf ihrer Grundlage realisiert. Die gewünschte Lernfähigkeit wird durch die Einführung einer neuen Knotenart im verwendeten Optionenbaum der DPA realisiert. In den folgenden Kapiteln wird diese Knotenart als RL-Choice-Option bezeichnet. Die um die Lernfähigkeit erweiterte DPA basiert auf dem Reinforcement-Learning-Ansatz. Die Bezeichnung RL-Choice stammt aus den beiden Tatsachen, dass die Wahl (engl.: choice) der möglichen Alternativen und das Lernen mit dem RL-Ansatz stattfindet. Der Lernprozess erfolgt auf der Ebene, auf der die abstrakten Aktionen umgeschaltet werden.
Die DPA-RL-Architektur beinhaltet den Planungsprozess, den Ausführungsprozess, die Informationsverwaltung (Optionenbaum) und Informationsvervollständigung (Lernen). In Abbildung 7.4 ist der auf Grundlage der DPA-RL-Architektur aufgebaute DPA-RL-Agent abgebildet. Der Planungsprozess trifft

eine Entscheidung darüber, welche abstrakte Aktion bzw. welcher Teil der abstrakten Aktion in der aktuellen Situation am besten zur aktuellen Situation passt. Der Ausführungsprozess ist die Propagierung der aktuellen Messungen durch die vorgegebene Datenstruktur, die als ein hierarchisch aufgebauter Baum repräsentiert und als *Optionenbaum* bezeichnet wird. Beide Prozesse sind durch rote und blaue Pfeile, die den Optionenbaum durchdringen, gekennzeichnet und laufen parallel ab. Gesucht ist eine Strategie, eine Kette der abstrakten Aktionen, die den Agenten in minimaler Zeit zum Ziel navigiert. Die gesuchte Strategie des Agenten wird Schritt für Schritt also sequenziell gebildet. Die gesuchte Strategie wird auf Grundlage der gemachten Erfahrungen verbessert, so dass die gierige Strategie, angewendet auf die erlernte Nutzenfunktion, nach einiger Zeit ein gutes Verhalten aufweist. Ein gutes Verhalten wird als schnelle Steuerung des Agenten zum Ziel aufgefasst.
Mithilfe des Ausführungsprozesses wird die gewünschte Reaktivität des Agenten implementiert. Die Entscheidung darüber, welche Aktionen in welchem Zustand möglich sind, übernimmt der Ausführungsprozess. Diese Strategie des Agenten steuert die Menge der abstrakten Aktionen, die im Optionenbaum verankert sind, und nimmt am Planungsprozess teil, wobei die mögliche Aktionsmenge im aktuellen Zustand, sowie der Zeitpunkt, wann die Entscheidung getroffen werden soll, von beiden Prozessen bestimmt werden.
Die abstrakten Aktionen sind miteinander über den RL-Choice-Knoten im vorgegebenen Optionenbaum verknüpft.
Die DPA-RL-Architektur ist durch ein Zusammenspiel aus der Vorbereitung und dem Aufbau der benötigten Informationen und dem Lernen nur an den wichtigen Stellen entstanden.

7.1.5 Literaturübersicht

Die Doppelpass-Architektur wird im RoboCup angewendet und orientiert sich an Multi-Agenten-Systemen. In [Berger et al., 2006] sind Erfolge der Anwendung der DPA beschrieben. Das Reinforcement-Learning ist auch in der Doppelpass-Architektur zum Einsatz gekommen, allerdings wird RL zum Erzielen der reaktiven Fähigkeiten (also Einsatz in der niedrigen Abstraktionsebene) angewendet, siehe [Berger, 2006, Kap. 2.3]. Der Unterschied der vorliegenden Arbeit zur DPA liegt in der Anwendung der RL-Methoden zur Bildung der Strategie, die mit abstrakten Aktionen operiert.
In der Arbeit [Koenig & Simmons, 1998] wird eine ähnliche Problemstellung behandelt. Es existiert ein autonomer Roboter, der in einer Umgebung mit Ungenauigkeiten interagiert. Eine topologische Karte der Umgebung ist vorhanden. In der Arbeit von Koenig und Simmons werden grundlegende Bestandteile eines Navigationsproblems behandelt: die Schätzung der Position des Roboters, Wegplanung, Kontrolle während der Navigation und Lernen. Die POMDP-basierte (POMDP: partially observable Markov decision process) Navigationsarchitektur verwendet die aus einer topologischen Karte automatisch erzielte POMDP, Aktuatoren und Sensormodelle und unbestimmtes Wissen aus der Umgebung. Ein POMDP wird aus Beobachtungen gelernt und angewendet, um die Wahrscheinlichkeitsverteilung der aktuellen Position des Roboters zu bestimmen. Der POMDP wird auch für die Bestimmung der Strategie des Roboters angewendet.
Die Grundidee, die in dieser Arbeit verfolgt wird, ähnelt der in [Cocora et al., 2006] angewendeten Idee. Dabei wird in der Arbeit von Cocora et al. das sogenannte relational reinforcement learning angewendet, um einen relationalen Entscheidungsbaum zu lernen. Dieser Entscheidungsbaum wird aus Sequenzen von besuchten Stellen erlernt, während der Roboter seine Aufgabe erfüllt. Dieser rationale Entscheidungsbaum repräsentiert die abstrakte Navigationsstrategie. Anschließend kann der erlernte Baum auch in anderen Szenarien angewendet werden, da viele Gebäude eine ähnliche Struktur aufweisen. In der Arbeit von Cocora et al. wird auch die Möglichkeit der Einbindung der RL-Algorithmen bei Navigati-

onsproblemen verfolgt. Die abstrakten Aktionen in dieser Arbeit werden durch komplexe Aufgaben, wie ”einen Flur durchfahren”, ”Fahr zum Ausgang des Zimmers”, u.a. beschrieben und sind von der Topologie des Raumes abhängig. Eine solche Vorgehensweise kann Vor- und Nachteile aufweisen. In der vorliegenden Arbeit wird statt relational reinforcement learning das hierarchische Reinforcement-Learning angewendet. Die abstrakten Aktionen in der vorliegenden Arbeit sind unabhängig von der Topologie der angewendeten Szenarien.
In [Hamdi, 1999] wurde für den Entwurf adaptiver, lernender Roboter eine ähnliche Idee wie im DPA-RL-Agenten verfolgt. Die Menge der Kontrollfunktionen wurde zusammengestellt, und die Aktivierung dieser Kontrollfunktionen in den vorgegebenen Zuständen wird mithilfe einer RL-Methode erlernt. Die verwendeten Kontrollfunktionen unterscheiden sich aber von den in der vorliegenden Arbeit angewendeten reaktiven Fähigkeiten.
Ein Hybrid aus Planung und Lernen in einem System ist zum Beispiel in der Arbeit von Ryan beschrieben [Ryan, 2002]. Dieses System basiert auch auf dem Ansatz des Hierarchical-Reinforcement-Learning. In der Arbeit von Ryan wird die gleiche Idee wie in der vorliegenden Arbeit verfolgt. Eine Strategie wird erlernt, die mit abstrakten Aktionen operiert. Eine topologische Karte liegt vor. Die abstrakten Aktionen (tele-operators) in [Ryan, 2002] haben Ähnlichkeiten mit den in der vorliegenden Arbeit angewendeten abstrakten Aktionen. Die tele-operators beinhalten sowohl Vorbedingungen als auch Nachbedingungen. Basierend auf der Topologie der Umgebung werden die abstrakten Aktionen aus den elementaren Aktionen zusammengesetzt. Diese Zusammensetzung wird auch gelernt. In diesem Punkt unterscheidet sich die vorliegende Arbeit von der Arbeit von Ryan, da dies einen zusätzlichen Aufwand bereitet und die zusätzlichen lokalen Sensoren nicht mitberücksichtigt werden, so dass viel Zeit aufgewendet wird, um eine Strategie zu erlernen, und die Anwendung in der Realität und im Onlinebetrieb nicht möglich ist.
In [Dorigo & Colombetti, 1998] werden ähnliche Ideen für das Lernen auf höherer Abstraktionsebene präsentiert. Die betrachteten Problemstellungen und mit dieser Problemstellung gebildeten abstrakten Aktionen, sowie die angewendete Architektur in diesem Buch unterscheiden sich von denen in der vorliegenden Arbeit. Die Ideen zu Hierarchical-Reinforcement-Learning sind zum Beispiel in [Dietterich, 2000b], [Dietterich, 2000a], [Sutton et al., 1999] ausführlich dargestellt. In [Stone, 1998] wurde das hierarchische Lernen mit dem Reinforcement-Learning-Ansatz für Multiagentensysteme eingeführt.

7.2 DPA-RL-Agent zur Roboternavigation

Zur besseren Verständlichkeit wird der DPA-RL-Agent mit anderen Agententypen verglichen. Der verwendete Optionenbaum zur Lösung des Navigationsproblems, die im Optionenbaum verwendeten Knotenarten, die Planungs- und Ausführungsprozesse und der Synchronisationsschritt werden im vorliegenden Kapitel betrachtet.

7.2.1 Vergleich des DPA-RL-Agenten mit anderen Agententypen

Im Folgenden werden die wichtigsten Merkmale der grundlegenden Architektur kurz beschrieben und ihre Besonderheit im Vergleich zu anderen, in der vorliegenden Arbeit verwendeten Agententypen und Architekturen verdeutlicht.

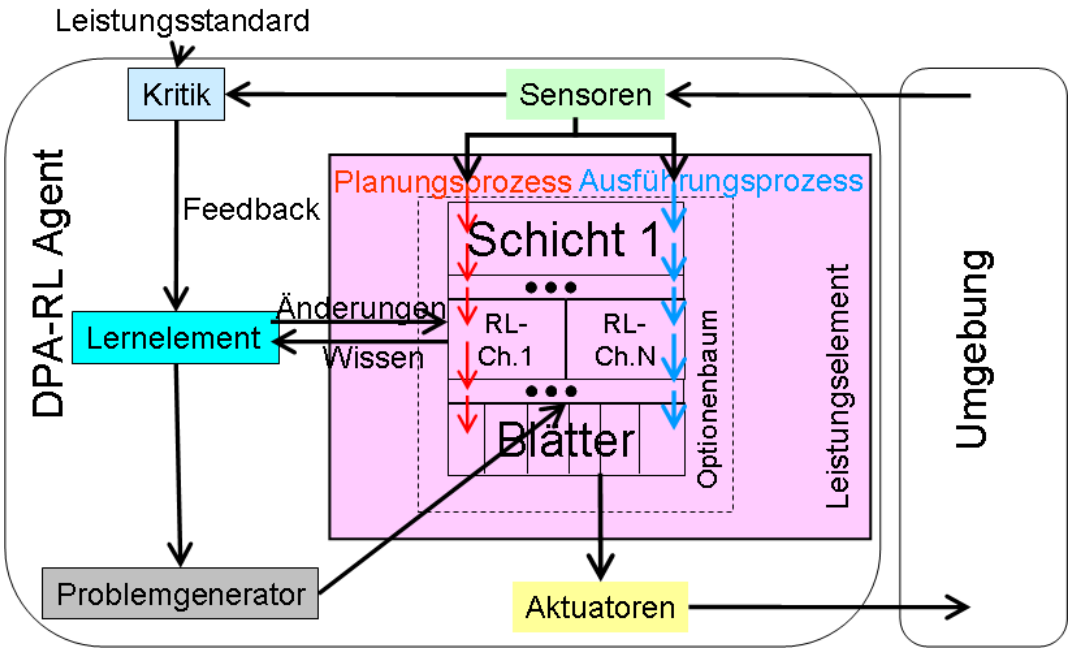


Abbildung 7.4: Darstellung des DPA-RL-Agenten im Sinne eines lernenden Agenten, siehe Abbildung 2.2. Die grundlegenden Bestandteile der lernenden Agenten sind in beiden Abbildungen mit gleichen Farben gekennzeichnet.

7.2.1.1 DPA-RL-Agent als lernender Agent

In Kapitel 2 ist der allgemeine Aufbau der rationalen, lernbasierten Agenten beschrieben. Der DPA-RL-Agent repräsentiert eine konkrete Realisierung dieses allgemeinen Konzepts. Der DPA-RL-Agent wird für die Gegebenheiten des Scorpion-Roboters und für die Lösung des Navigationsproblems realisiert. Die grundlegenden Bestandteile eines rationalen, lernenden Agenten bzw. des DPA-RL-Agenten sind in Abbildung 2.2 bzw. in Abbildung 7.4 durch gleiche Farben gekennzeichnet, vergleiche Kritik-Komponente, Lernelement, Leistungselement, Problemgenerator.
Das Leistungselement ist in beiden Abbildungen mit der Farbe rosa gekennzeichnet. Es wird im DPA-RL-Agent mithilfe der DPA-RL-Architektur realisiert. Das Leistungselement beinhaltet den Optionenbaum und zwei Kontrollprozesse, den Planungs- und Ausführungsprozess, siehe Abbildung 7.4. Für den RL-Knoten wird das interne Umgebungsmodell aufgebaut. Der interne Zustand, der aus den Sensoren extrahiert wird, besteht sowohl aus lokalen, aus IR-Sensoren ausgelesenen Informationen, als auch aus globalen, aus Odometriedaten ausgelesenen Informationen. Dieser interne Zustand wird von beiden Kontrollprozessen und vom Lernschritt angewendet. Das Lernelement beinhaltet den Anpassungsschritt der Nutzenfunktion des RL-Knoten an die aktuellen Messungen und wird durch RL-Methoden realisiert.
Die Kritik-Komponente ist mit der Farbe hellblau in beiden Abbildungen gekennzeichnet. Sie wird im DPA-RL-Agenten durch die Bestimmung des Belohnungsmodells realisiert. Mithilfe des Belohnungsmodells hat der Agent in bestimmten Situationen eine Vorstellung von der Güte der von ihm unternommenen abstrakten Aktionen.
Der Teil Problemgenerator, siehe graues Kästchen in Abbildung 7.4, wird durch Anwendung des ϵ-Anteils in der ϵ-gierigen Strategie realisiert, so dass mit vorgegebenem Zufall ϵ eine zufällige abstrakte Aktion gewählt wird. Dieser Bestandteil des rationalen lernbasierten Agenten ist für den Explorationsschritt verantwortlich und wird in [Russell & Norvig, 2004] als Problemgenerator bezeichnet.

7.2.1.2 DPA-RL-Architektur versus RL-Agent

Die neue Knotenart RL-Choice-Knoten im Optionenbaum beinhaltet den gleichen Lernalgorithmus wie der RL-Agent. Die angewendete Lernregel, die Bildung des TD-Fehlers, die Eligibility-Traces, das Konzept der Q-Funktion und das Lernen dieser Q-Funktion bleiben ähnlich, wie in Kapitel 3 beschrieben. Der RL-Choice-Knoten hat eine andere Aktionsmenge als der RL-Agent in Kapitel 5.1.2. Die im RL-Choice-Knoten verwendete Menge der abstrakten Aktionen wird in Abhängigkeit von der aktuellen Situation formuliert. Der Zustandsraum wird für den Bedarf des angewendeten Aktionsraumes dynamisch ausgebaut. Der so aufgebaute Zustandsraum enthält deutlich weniger Zustände als der des RL-Agenten. Die komplexere Verwaltung und die Erkennung, an welcher Stelle die Entscheidung wirklich notwendig ist, wird im DPA-RL-Agenten von den beiden Kontrollprozessen übernommen.
Beim RL-Agenten werden elementare Aktionen angewendet, die den Agenten entweder nur nach vorne bringen (z. B. $10\ cm$) oder den Agenten nach rechts oder links drehen (z. B. $-/+10°$). Den Zustandsraum kann man sich als ein kariertes Blatt mit $10\ cm \times 10\ cm$ großen Kästchen für jede mögliche Orientierung vorstellen. Solch ein Aufbau des Zustandsraumes beinhaltet mehrere Tausend Zustände und benötigt Hunderte von Episoden für das Erlernen der optimalen Strategie.

7.2.1.3 DPA-RL-Architektur versus DPA

Die DPA-RL-Architektur beinhaltet alle Bestandteile der DPA. Das Lernen des DPA-RL-Agenten erfolgt im Onlinebetrieb. Um lernen zu können, müssen einzelne Alternativen der Reihe nach, also nicht alle gleichzeitig, ausprobiert werden. Bei fehlenden Alternativen nach dem Abbruch der aktuellen Aktion wird der *Redeliberationsschritt* ausgeführt. Dieser Schritt ist in der grundlegenden DPA vorhanden. Im DPA-RL-Agent ist dieser Schritt ein wesentlicher Bestandteil der Architektur. Der Redeliberationsschritt ist eine Art Synchronisation beider Prozesse und ist notwendig für den Adaptionsschritt im RL-Choice-Knoten, siehe auch Unterkapitel 7.2.5.
Die Ähnlichkeit des RL-Choice-Knotens und des Choice-Knotens liegt darin, dass diese Knotenart die beste Alternative zwischen den vorgegebenen Möglichkeiten nach aktuellem Informationsstand wählt. Der Unterschied liegt in der Anpassung der Prioritätenfunktion, die im RL-Choice-Knoten als Nutzenfunktion realisiert und mit dem RL-Ansatz an die aktuellen Messungen adaptiert wird. Der Choice-Knoten bzw. RL-Choice-Knoten wird in Kapitel 7.2.3 bzw. in Kapitel 7.3.1 genauer erläutert.

7.2.2 Optionenbaum (Datenstruktur)

Der Optionenbaum beinhaltet das am Anfang vom Entwickler vorgegebene Wissen und eine hierarchisch aufgebaute Baumstruktur. Die Schichten dieses Baumes repräsentieren die unterschiedlichen Abstraktionsebenen eines Navigationsproblems. Ein Pfad im Optionenbaum von der Wurzel bis zum Blatt bildet einen Plan des DPA-RL-Agenten. Die Blätter des Optionenbaums, aus denen die abstrakten Aktionen zusammengesetzt sind, liegen auf der niedrigsten Ebene des Optionenbaums. In der Wurzel des Optionenbaums wird das Problem des höchsten Abstraktionsgrades definiert.
Das im Optionenbaum versteckte Wissen wird im RL-Choice-Knoten mit der RL-Methode während der Ausführung der Mission mit den aktuellen Messungen automatisch aktualisiert. Die DPA-RL-Architektur beinhaltet eine weitere Struktur zum Lernen. Während der Ausführung der Mission können neue abstrakte Aktionen hinzugefügt werden, so dass die Struktur des Baumes dynamisch erweitert wird. Nicht nur der ”Inhalt des Knotens” im Baum, sondern auch die Baumstruktur können verändert werden. Dieser Mechanismus wird in Kapitel 7.4 genauer erläutert.

Das in der vorliegenden Arbeit betrachtete Navigationsproblem beinhaltet nur ein ausgewähltes Ziel. Der Optionenbaum, in dessen Wurzel der RL-Choice-Knoten liegt und dessen Kindknoten die vorhandenen, abstrakten Aktionen sind, wird für die Lösung des Navigationsproblems angewendet. Die Mission des Agenten wird mehrere Male wiederholt, so dass der Agent in jeder Episode sein Verhalten verbessern kann, um nach ein paar Episoden eine gute Strategie zu erlernen.
In Abbildung 7.5 ist ein Beispiel für einen angewendeten Optionenbaum dargestellt. Die von Anfang an

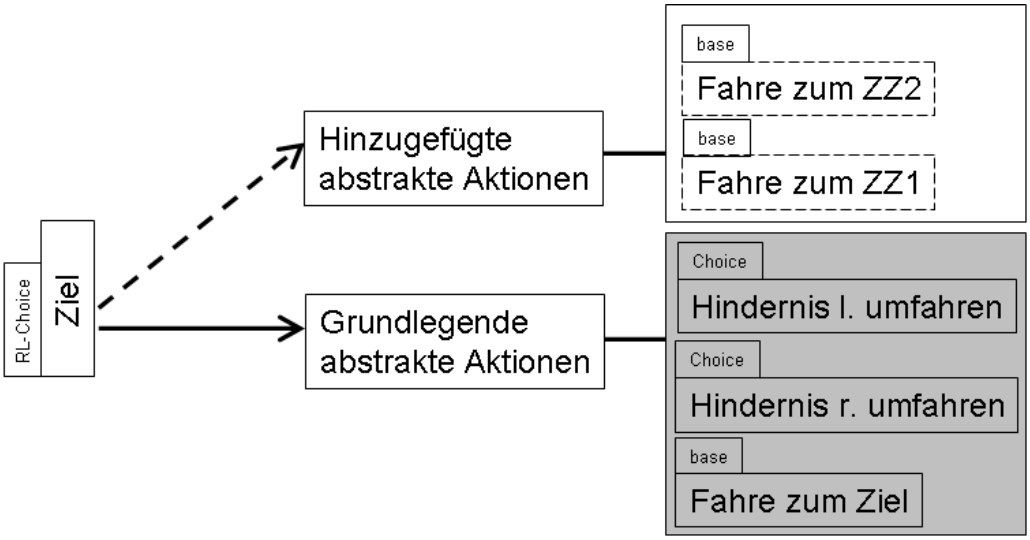


Abbildung 7.5: Optionenbaum für den DPA-RL-Agenten

vorhandenen, abstrakten Aktionen sind als ”grundlegende abstrakte Aktionen” mit einem grauen Kasten gekennzeichnet. Der Aktionsraum der RL-Choice-Knoten (Kindknoten) besteht aus drei abstrakten Aktionen ”Hindernis rechts umfahren” bzw. ”Hindernis links umfahren” und ”zum Ziel fahren”. Die durch Ausführung der Mission und durch Fortsetzung des Lernprozesses neu bestimmten, abstrakten Aktionen sind als ”hinzugefügte abstrakte Aktionen” in der Abbildung aufgeführt. Die Knotenart (RL-Choice-, Choice- oder Basis-Knoten) ist durch einen Kasten oberhalb des Knotens gekennzeichnet.
Jede Ebene des Optionenbaums hat einen Zugriff auf die Wahrnehmung und das interne Umgebungsmodell. Das Ziel wird durch einen Punkt im Raum in Bezug auf die Startposition des Agenten vorgegeben. Aus den vorgegebenen Zielen (hier Positionen) und Teilzielen wird die abstrakte Aktion ”zum Ziel fahren” generiert.

7.2.3 Knotenarten im Optionenbaum

Jede Knotenart im Optionenbaum beinhaltet die gleiche Struktur, die aus der Vorbedingung, der Nachbedingung und der Abbruchbedingung besteht. Diese Bedingungen werden für die Realisierung der Planungs- und Ausführungsschritte verwendet. Die Bedingungen werden auf Grundlage der aktuellen Messungen geprüft.

7.2.3.1 Knotenstruktur

Die *Vorbedingung* wird vom Planungsprozess angewendet. Der Planungsprozess läuft von oben nach unten durch den Optionenbaum (Top-Down-Prozess). In jeder Ebene des Optionenbaums entscheidet sich

der Planungsprozess aus einer oder mehreren Alternativen (in der aktuellen Situation mögliche, ausführbare Kindknoten) für den besten oder allein möglichen Kindknoten und steigt eine Ebene herunter. Die Vorbedingungen der Kindknoten geben die Information, ob die Alternative (Kindknoten) in der aktuellen Situation ausführbar ist. Zum Beispiel werden in der RL-Choice-Option die Vorbedingungen für die Bestimmung der in der aktuellen Situation möglichen Teilmenge der abstrakten Aktionen angewendet.
Die *Abbruchbedingung* und die *Nachbedingung* werden vom Ausführungsschritt geprüft um die Reaktivität des Systems gewährleisten zu können.
Die Nachbedingungen eines Knotens sind dann erfüllt, wenn die durch diesen Knoten definierten Ziele erfüllt sind, also ein Teilschritt zum globalen Ziel gemacht ist und weitere Schritte zum verfolgten Ziel des Vaterknotens gemacht werden können.
Die Abbruchbedingungen werden für die Realisierung der Reaktivität verwendet. Das durch den Planungsprozess ausgewählte Blatt im Optionenbaum wird nur ausgeführt, wenn es möglich ist. Wenn die verfolgte Aktion bzw. Teilaktion die Abbruchbedingungen erfüllt, wird der Ausführungsprozess abgebrochen. In diesem Fall können alternative Pläne, die im Planungsprozess schon berechnet sind, verfolgt werden[40]. Wenn keine alternativen Pläne vorhanden sind, wird der sogenannte Redeliberationsschritt gemacht und ein neuer Plan durch den Planungsprozess aufgestellt, siehe auch Unterkapitel 7.2.5.
Der Unterschied der Knotenarten liegt im Auswahlmechanismus der Kindknoten, der wichtig für den Planungsprozess ist. Im Optionenbaum der DPA-RL-Architektur existieren vier unterschiedliche Knotenarten: Choice-Knoten (Auswahlknoten), RL-Choice-Knoten, Sequenzknoten und die Basis-Knoten (Blätter). Der Ausführungsprozess behandelt alle Knotenarten gleich.

7.2.3.2 Knotenarten

Die *Basis-Knoten* beinhalten keine Kindknoten und benötigen keinen Mechanismus zur Auswahl der möglichen Kindknoten. Ein Basis-Knoten meldet sich bei seinem Vaterknoten, wenn er nicht mehr ausführbar ist. Dieses wird durch die Abbruchbedingung organisiert. Der Basis-Knoten ist ein ausführbarer Knoten, der Abbruch- und Vorbedingungen beinhaltet. Diese Knotenart beinhaltet keine Nutzenfunktion.
Der *Choice-Knoten*, also der Auswahlknoten, beinhaltet die Fähigkeit die möglichen Alternativen auszuwählen. Es gibt zwei Varianten des Choice-Knotens. Die erste Variante beinhaltet eine Prioritätenfunktion, die die möglichen Alternativen unter aktuellen Randbedingungen auswertet. Die zweite Variante realisiert das Ausschlussverfahren, entweder wird eine bestimmte Alternative oder eine andere übernommen, je nach aktuellen Messungen. Diese Variante verknüpft die Kindknoten mit einem nichtexklusiven ”Oder” miteinander. Diese Art der Knoten ist der Spezialfall der ersten Variante. Je nach Randbedingungen ist die eine oder eine andere Alternative besser oder gar die einzige Möglichkeit.
Ein *Sequenzknoten* beinhaltet als Kindknoten eine Reihe von auszuführenden Optionen, die der Reihe nach ausgeführt werden müssen, um das vom Vaterknoten verfolgte Ziel zu erreichen. Im Vaterknoten wird die Reihenfolge (keine Prioritätenliste) für Kindknoten bestimmt. Die Erfüllung der Nachbedingung eines Kindknotens ist die Erfüllung der Vorbedingung für den nächsten Kindknoten. Die Kindknoten der Sequenzknoten sind die Teilschritte des durch den gewählten Sequenzknoten verfolgten Plans. Der Sequenzknoten hat eine feste Anordnung der auszuführenden Teilschritte. Kein Teilschritt kann ohne andere Teilschritte ausgeführt werden. Mithilfe dieser Knotenart kann zum Beispiel eine Inspektionsaufgabe formuliert werden. Diese Art der Knoten ist sehr wichtig im Bereich der Fußball-Roboter. Der in der vorliegenden Arbeit angewendete Optionenbaum beinhaltet diese Knotenart nicht.
Der *RL-Choice-Knoten* wird aus dem Choice-Knoten abgeleitet. Er beinhaltet auch eine Prioritätenfunktion, die angibt welche der möglichen Alternativen (Kindknoten) in der aktuellen Situation den maxi-

[40] Dieser Fall tritt in der Standard-DPA auf.

malen Nutzen verspricht. An der Stelle, wo die Nutzenfunktion definiert wird, wird die Möglichkeit ausgeschöpft, den Lernmechanismus einzubauen. Diese Nutzenfunktion wird mit der Reinforcement-Lernregel an die aktuelle Messung (aus dem Feedback der Umgebung) angepasst.

7.2.4 Ausführungs- und Planungsprozesse für das Navigationsproblem

Im DPA-RL-Agenten findet der Lernprozess statt, die Nutzenfunktion soll an die aktuellen Messungen angepasst werden. Aus diesem Grund sollen beide Kontrollprozesse in der DPA-RL-Architektur nacheinander ein- und ausgeschaltet werden, also synchronisiert werden. In der DPA können beide Prozesse unabhängig voneinander laufen, so dass während der Ausführungsphase die nächsten alternativen Schritte vorgeplant werden können.
Beide Prozesse werden über die in allen Knotenarten verankerten Vor-, Nach- und Abbruchbedingungen realisiert. Der Planungsprozess kann nur die Zweige des Optionenbaums in Betracht ziehen, die die Vorbedingung erfüllen. Mithilfe der Vorbedingungen der Kindknoten wird die Menge der in der aktuellen Situation möglichen, ausführbaren, abstrakten Aktionen der RL-Choice-Knoten gebildet.

7.2.4.1 Realisierung der Planungs- und Ausführungsprozesse

Die Erstellung des Plans für die aktuelle Situation übernimmt der Planungsprozess. Durch den Abstieg im hierarchisch aufgebauten Baum wird zuerst ein grobes Konzept auf den höheren Ebenen des Optionenbaums aufgestellt und erst danach mit weiterem Abstieg zu den Blättern des Baumes eine detailliertere Betrachtung der Aufgabe vorgenommen. Somit wird zuerst die grobe Richtung vorgegeben und erst dann Schritt für Schritt der Plan unter Berücksichtigung der aktuellen Informationen, die aus Odometriedaten und IR-Sensoren gewonnen werden, verfeinert. Eine solche Vorgehensweise ist auch für den Menschen typisch.
In Abhängigkeit vom verfolgten Ziel und aktuellen Messungen wird nach aktuellem Wissensstand der beste Plan (Weg von der Wurzel bis zum Blatt) durch den Planungsprozess vorgeplant. Dabei wird auf die Ausführbarkeit der einzelnen Schritte geachtet.
Der durch den Planungsprozess vorgeplante Weg im Optionenbaum wird auf jeder Ebene des Optionenbaums einschließlich der Blätter durch Umschalten der Knoten auf den Status ”Intention” realisiert. Der Ausführungsprozess geht über den vorgeplanten Weg über die Knoten, die sich im Status ”Intention” befinden, bis zum Blatt hinab.

7.2.4.2 Ablaufbeispiel für das Navigationsproblem

Wenn der Agent im Navigationsproblem am Anfang keine Hindernisse auf dem Weg zum Ziel spürt, wird vom Planungsprozess die Aktion ”zum Ziel fahren” ausgewählt. Dieser Kindknoten wird in den Status ”Intention” umgeschaltet. Die Kontrolle wird an den Ausführungsprozess übergeben. Die gewählte abstrakte Aktion ”zum Ziel fahren” wird solange ausgeführt, bis sie nicht mehr ausführbar ist. Das ist zum Beispiel dann der Fall, wenn auf dem Weg zum Ziel ein Hindernis steht, das die direkte Fahrt zum Ziel versperrt. In diesem Fall wird die Abbruchbedingung für die gewählte abstrakte Aktion erfüllt sein. Wenn der Kindknoten der RL-Choice-Option die Abbruchbedingungen erfüllt, wird dieser Knoten vom Status ”Intention” auf den Status ”abgebrochen” umgeschaltet. Dann wird die Aktion als beendet erkannt, der Agent wird in den neuen Zustand überführt.

7.2.4.3 Aspekte der Implementierung der Vor- und Abbruchbedingungen

Die Kindknoten der RL-Choice-Optionen werden als abstrakte Aktionen bezeichnet. Wenn die gewählte abstrakte Aktion mehrere Teilschritte (Kindknoten) beinhaltet, ist es wichtig, dass eine der Abbruchbedingungen dieser abstrakten Aktionen von den Vorbedingungen der Kindknoten abhängig ist.
Die Vorbedingungen werden auf Grundlage der Informationen definiert, zu denen alle Schichten einen Zugriff haben, so dass die Vorbedingungen des Vaterknotens in Abhängigkeit von den Vorbedingungen des Kindknotens definiert werden können, ohne dass die Kontrollprozesse bis zum Kindknoten absteigen müssen. Die Kindknoten repräsentieren die Teilschritte des Vaterknotens. Wenn die Abbruchbedingung des Teilschrittes (Kindknoten) erfüllt ist, wird der Status des Vaterknotens vom Status ”Intention” in ”abgebrochen” umgeschaltet und der Planungsprozess vom Ausführungsprozess aufgerufen.
Der Planungsprozess läuft den vorgeplanten Weg, der durch Zustände mit dem Status ”Intention” markiert ist, von der Wurzel bis zu der Stelle, wo die Entscheidung getroffen werden soll, um den Plan zu vervollständigen.
Wenn der RL-Choice-Knoten im Status ”Intention” ist, wird der Planungsprozess bis zur gewählten abstrakten Aktion steigen. Wenn die gewählte abstrakte Aktion mehrere Teilschritte beinhaltet, wird entschieden welcher der Teilschritte aufgerufen werden soll. Wenn der RL-Choice-Knoten im Status ”abgebrochen” ist, wird zuerst der Lernprozess und danach der Planungsprozess aufgerufen. Der Planungsprozess bestimmt mithilfe der aktuellen Q-Funktion und darauf angewendeten ϵ-gierigen Strategie die beste abstrakte Aktion in der aktuellen Situation. Diese beste abstrakte Aktion wird für die aktuelle Situation aus den möglichen, ausführbaren, abstrakten Aktionen, also den Kindknoten, deren Vorbedingungen erfüllt sind, ausgewählt. Jede Aktion kann in jedem Moment in den Status ”abgebrochen” umgeschaltet werden, und so wird die Reaktivität des Systems erreicht.

7.2.5 Synchronisation der Planungs- und Ausführungsprozesse

Der Schritt der Anpassung der Nutzenfunktion benötigt einen Synchronisationsschritt zwischen dem Ausführungsprozess und dem Planungsprozess, da ohne die Meldung des Ausführungsprozesses und des Übergangs in den neuen Zustand kein Anpassungsschritt der Nutzenfunktion stattfinden kann, und ohne die Entscheidung über die nächste Aktion im neuen Zustand keine Ausführung der neu bestimmten Aktion stattfinden kann. Aus diesem Grund wird der Redeliberationsprozess in jedem Schritt der RL-Choice-Knoten durchgeführt. Der Redeliberationsprozess ruft die Aktualisierung der Nutzenfunktion auf und setzt den Planungsprozess fort. Der Planungsprozess bestimmt die neue Aktion im neuen Zustand, und dies vervollständigt den benötigten Plan, so dass der Ausführungsprozess fortgesetzt werden kann. Diese beiden Teilschritte, Lernschritt und Bestimmung der neuen Aktion, werden in getrennten Funktionen eingekapselt, um eine bessere Flexibilität des Gesamtsystems gewährleisten zu können.
Mithilfe des Redeliberationsprozesses werden beide Kontrollprozesse synchronisiert. Und die Entscheidungen erfolgen auf Grundlage der aktualisierten Nutzenfunktion in Echtzeit.

7.3 Lernfähigkeit des DPA-RL-Agenten

In diesem Kapitel wird eine neue Knotenart, der RL-Choice-Knoten, eingeführt. Der Aufbau der Kindknoten des RL-Choice-Knotens (vom Agenten benötigte abstrakte Aktionen) und der Aufbau des Zustandsraumes werden auch in diesem Kapitel beschrieben. Diese Bestandteile der Aktionsmenge und des Zustandsraums sind substantiell für den Lernprozess.

Dem Belohnungsmodell wird wenig Aufmerksamkeit gewidmet. Im DPA-RL-Agent wird das gleiche Belohnungsmodell wie im RL-Agent verwendet.

7.3.1 RL-Choice-Knoten

Im RL-Choice-Knoten soll die optimale Strategie, die aus einer Kette von abstrakten Aktionen besteht und den Agenten auf dem schnellsten Weg zum Ziel führt, erlernt werden. Die Strategie des Agenten bildet den aktuellen Zustand des RL-Choice-Knotens auf die bestmögliche, abstrakte Aktion in der aktuellen Situation (im aktuellen Zustand) ab.
Aus der Nutzenfunktion (hier Q-Funktion) wird die Strategie des Agenten abgeleitet. Die Nutzenfunktion gibt die Priorität der möglichen Aktionen im aktuellen Zustand an.

7.3.1.1 Lernschritt im RL-Choice-Knoten

Ein Zeitschritt im Gesamtsystem unterscheidet sich von einem Lernschritt, der in der RL-Choice-Option abläuft. Eine abstrakte Aktion kann mehrere Zeitschritte, die im System ablaufen, dauern. Die unterschiedlichen, abstrakten Aktionen benötigen eine unterschiedliche Anzahl an Zeitschritten des Systems. Der Lernprozess der DPA-RL-Architektur wird durch den SARSA-Algorithmus realisiert und läuft im RL-Choice-Knoten ab. Ein Lernschritt[41] wird dann durchgeführt, wenn der RL-Choice-Knoten vom alten Zustand s in den neuen Zustand s' überführt wird. Das System wird in den neuen Zustand überführt, wenn die gewählte, abstrakte Aktion die Abbruchbedingungen erfüllt, und der Planungsprozess eine neue Entscheidung über den besten Kindknoten der RL-Choice-Option, also die Entscheidung für die beste abstrakte Aktion, treffen muss.
Die abstrakte Aktion muss dann abgebrochen werden, wenn eine Gefahr für den DPA-RL-Agenten entsteht oder die ”Geduld” des Agenten ausgeschöpft ist. Die Gefahr wird mithilfe der Abbruchbedingungen erkannt. Die abstrakte Aktion wird als beendet deklariert, und der RL-Choice-Knoten wird in den neuen Zustand überführt, wenn die Abbruchbedingung der ausführbaren, abstrakten Aktion erfüllt ist. Zum Beispiel, wenn sich ein Hindernis vor dem Agenten befindet, und die abstrakte Aktion ”zum Ziel fahren” nicht mehr ausführbar ist, wird die Abbruchbedingung für den Kindknoten ”zum Ziel fahren” wahr sein. Die Synchronisation beider Kontrollprozesse ist sehr wichtig. Wenn die Abbruchbedingung der durch den Planungsprozess bestimmten abstrakten Aktion gilt, kann der Ausführungsprozess nicht weiter ablaufen. Der RL-Choice-Knoten wird in den neuen Zustand überführt und der Lernschritt durchgeführt. Anschließend wird der Planungsprozess zur Bestimmung der neuen Aktion aufgerufen. Diese drei Bestandteile beinhaltet der Redeliberationsschritt.

7.3.1.2 Lernprozess im RL-Choice-Knoten

Das Lernen im RL-Choice-Knoten erfolgt wie bei allen RL-Methoden durch den Prozess der Generalized-Policy-Iteration. In Kapitel 3.2.3 sowie [Sutton & Barto, 1998, Kap. 4.6] ist dieser Prozess grob beschrieben. Dieser Prozess besteht aus zwei Bestandteilen: Anpassung der Q-Funktion konsistent zur aktuellen Strategie (engl.: policy evaluation) und Herstellung der gierigen Strategie in Hinsicht auf die aktuelle Q-Funktion (engl.: policy improvement). Diese beiden Schritte werden für die Annäherung an die optimale Strategie und damit verbundene, optimale Q-Funktion benötigt. Die Evaluation der Strategie wird aufgerufen, wenn eine Aktion ausgeführt worden ist, der nächste Zustand erfasst ist, und die

[41] die Adaption der Nutzenfunktion an die aktuelle Messung

Belohnung aus dem Kritik-Komponente mitgeteilt worden ist. Die Anpassung der Nutzenfunktion an die aktuelle Messung wird auch als Lernschritt bezeichnet. Die Entscheidung über die nächste abstrakte Aktion und die Ausführung der getroffenen Entscheidungen bilden den Schritt Verbesserung der Strategie. Die beiden Schritte sind zwei parallel ablaufende Prozesse, die unabhängig voneinander verarbeitet werden können.
Der Lernschritt wird nach der Ausführung der gewählten Aktion erfolgen. Die Aktion wird mithilfe der ϵ-gierigen Strategie bestimmt, und in den meisten Fällen wird die Aktion, die die maximale Belohnung verspricht, dann als ausführbare Aktion an den Ausführungsprozess übergeben. Die Anpassung der Q-Funktion erfolgt erst nach dem Übergang des RL-Choice-Knotens in den neuen Zustand, im Redeliberationsschritt.
Der Ausführungsprozess meldet die Änderung des Zustandes im System, so dass der neue Zustand bestimmt werden kann, und die durchgeführte Aktion im alten Zustand bewertet werden kann. Mithilfe des Lernschrittes, siehe Gleichung (7.1), wird die Bewertung der ausgeführten Aktion im Hinblick auf das Erreichen des globalen Zieles vorgenommen. Durch Anpassung der Q-Funktion wird die aus der neuen Q-Funktion abgeleitete Strategie des Agenten angepasst. Der Planungsprozess entscheidet auf Grundlage der aktuellen Q-Funktion und möglichen Aktionsmenge, durch Anwendung der ϵ-gierigen Strategie, über den nächsten Schritt.
Nach der Meldung des Ausführungsprozesses, dass das System im neuen Zustand ist, befindet sich der Agent im Zustand s'. Dem Agenten sind der vorherige Zustand s, die ausgeführte Aktion a sowie die Entscheidung der ϵ-gierigen Strategie über die nächste Aktion a' bekannt. Die zur Ausführung der vorherigen Aktion benötigte (System-)Zeit T in Zeitschritten des Gesamtsystems, und das in diesem Zeitabschnitt empfangene Belohnungssignal $r = r_1 := r_2 := ... := r_T$ sind ebenfalls vorhanden. Mit dieser Information kann der Adaptionsschritt für die gesuchte optimale Q-Funktion im RL-Choice-Knoten mithilfe der modifizierten Lernregel für das Semi-Markow-Entscheidungsproblem [Parr, 1998, Kap. 3.2, S. 38] durchgeführt werden, vergleiche auch Kapitel 3.5.

$$Q^{neu}(s,a) := Q(s,a) + \alpha \cdot (r + \gamma^T \cdot Q(s',a') - Q(s,a)) \tag{7.1}$$

Die nächste Aktion $a' \leftarrow \epsilon - Gierig_{a \in A(s')} Q(s', \cdot)$ wird auf Grundlage des aktuellen Zustands und der Aktionsmenge bestimmt. Der aktuelle Zustand s', der aus Odometriedaten des Roboters bestimmt wird, und die durch den Ausführungsprozess gebildete, aktuelle Aktionsmenge $A(s')$ werden benötigt, um diesen Schritt durchführen zu können. Die abstrakten Aktionen, die die Vorbedingungen erfüllen, werden zur aktuellen Aktionsmenge $A(s')$ hinzugefügt und beinhalten somit die in der aktuellen Situation möglichen abstrakten Aktionen. Der Zustandsraum, der zur Bildung der Q-Funktion benötigt wird, wird inkrementell gebildet.

7.3.2 Zustandsraum

Der DPA-RL-Agent baut seinen Zustandsraum inkrementell je nach Bedarf auf. Der Prozess des Aufbaus des Zustandsraumes wird im RL-Choice-Knoten realisiert. Nachdem das System einen Übergang in den neuen Zustand $s' \leftarrow \tau(s,a)$ meldet (Redeliberationsschritt) wird zuerst überprüft, ob der neue Zustand s' schon vom Agenten angefahren worden ist oder wirklich neu ist. Wenn der Zustand s' im Zustandsraum Z unbekannt ist, wird er mit seiner lokalen Umgebung $M_{s'}$ in der Liste gespeichert $Z \leftarrow Z \cup \{s', M_{s'}\}$. Die Zustände werden der Reihe nach gespeichert. Nachbarzustände sind die Zustände, die durch eine gefahrene Strecke verbunden sind, so dass im Zustandsraum Z die Verbindungen zwischen den Zuständen latent vorhanden sind. Der Aufbau des Zustandsraumes wird im Unterkapitel 7.4.1 genauer erläutert.

Die aufgebaute Liste beinhaltet sowohl die Menge der besuchten Zustände als auch die in jedem Zustand

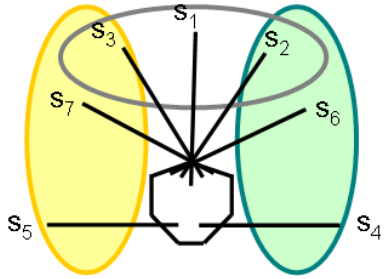
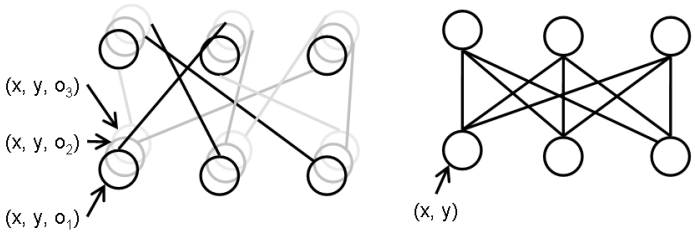


Abbildung 7.6: Bereiche zur Bestimmung des Abstands vom Agenten bis zum Hindernis. Der vordere Bereich ist durch eine graue Ellipse gekennzeichnet und wird über drei nach vorne ausgerichtete Sensoren (s_1, s_2, s_3) kontrolliert. Der Bereich auf der rechten Seite wird durch die drei zur rechten Seite ausgerichteten Sensoren (s_2, s_4, s_6) kontrolliert. Der Bereich auf der linken Seite wird durch die drei zur linken Seite ausgerichteten Sensoren (s_3, s_5, s_7) kontrolliert.

ermittelte, lokale Umgebung dieser Zustände. Die Daten aus den IR-Sensoren werden in jedem Zustand s diskretisiert und geben für den Zustand s an, ob das Hindernis in diesem Zustand links, rechts oder geradeaus, nah, mittelweit, weit oder gar nicht spürbar war.
Die lokale Umgebung im Zustand s besteht aus der Beschreibung der Lage der Hindernisse für den Agenten sowie der Lage des Ziels ($H_{vorne,\{nah,mittel,weit\}}(s) \in \{false, true\}$, $H_{rechts,\{nah,mittel,weit\}}(s) \in \{false, true\}$, $H_{links,\{nah,mittel,weit\}}(s) \in \{false, true\}$, $Ziel_{\{nah,mittel,weit\}}(s) \in \{false, true\}$).
Die Lage der Hindernisse wird aus den IR-Sensoren ermittelt.
In Abbildung 7.6 sind drei Bereiche für die Bestimmung des Wertes
($H_{vorne,\{nah,mittel,weit\}}(s)$, $H_{rechts,\{nah,mittel,weit\}}(s)$, $H_{links,\{nah,mittel,weit\}}(s)$) gekennzeichnet. Der Bereich vor dem Roboter-Agenten wird durch drei nach vorne ausgerichtete Sensoren s_1, s_2, s_3 kontrolliert. Aus dem nach vorne gerichteten Sensorwert s_1, sowie aus den auf den Strahl s_1 projizierten Sensorwerten ${s_2}^{s_1,'}$ und ${s_3}^{s_1,'}$ wird der kleinste Wert ermittelt $d_{vorne} \leftarrow min\{s_1, {s_2}^{s_1,'}, {s_3}^{s_1,'}\}$. Ob sich das Hindernis vor dem Agenten befindet und wenn ja, in welchem Abstand $\{nah, mittel, weit\}$, wird mithilfe des Wertes d_{vorne} bestimmt. Dasselbe Prinzip gilt für die rechte und linke Seite des Roboters.
Die Zielposition wird in einem Koordinatensystem mit dem Ursprung in der Startposition und der Orientierung des Roboters vorgegeben. Auf Grundlage des ermittelten Abstands bis zum Ziel wird der Wert $Ziel_{\{nah,mittel,weit\}}(s) \in \{false, true\}$ berechnet.
An dieser Stelle wird die auf Grundlage dieses Zustandsraumes Z abgeleitete Struktur *Graph'* eingeführt. Diese Struktur ist für die Bestimmung der neuen abstrakten Aktionen wichtig. $Graph' := \{K, V\}$ ist die Projektion des Zustandsraums Z, dabei werden alle Zustände $s \in Z$ mit gleicher Position und unterschiedlicher Orientierung zu einem Knoten K im Graphen $Graph'$ zusammengefasst, siehe Abbildung 7.7. Alle Verbindungen, die aus dem Zustandsraum Z extrahiert werden, werden im neuen Graphen übernommen, vergleiche Abbildung 7.7. Die wiederholten Verbindungen werden aus der Kantenmenge V von $Graph'$ weggestrichen.

7.3.3 Abstrakte Aktion "zum Ziel fahren" als Basis-Knoten

Für das zu lösende Navigationsproblem existiert eine vorgegebene, globale Zielposition. Die abstrakte Aktion "zum Ziel fahren" steuert den Roboter auf den vorgegebenen Zielpunkt zu. Die Umrechnungsschritte und die für die Regelung benötigten Werte sind in Kapitel 6.1.2 erläutert.



(a) Zustandsraum (b) Graph'

Abbildung 7.7: Projektionsschritt vom Zustandsraum Z auf den Graphen $Graph'$. Die Zustände in $Graph'$ unterscheiden sich nicht im Orientierungsanteil des Zustandes. Alle Zustände mit gleicher Position und unterschiedlicher Orientierung werden zu einem Knoten in $Graph'$ zusammengefasst. Alle Verbindungen, die im Zustandsraum vorhanden sind, werden in $Graph'$ übernommen.

Der Kindknoten "zum Ziel fahren" des RL-Choice-Knotens ist eine der vorgegebenen abstrakten Aktionen. Dieser Kindknoten ist als ein Blatt (Basis-Knoten) im Optionenbaum realisiert.
Die Vorbedingung für den Basis-Knoten "zum Ziel fahren" ist erst erfüllt, wenn das Ziel nicht erreicht ist, und vorne vor dem Agenten keine Hindernisse vorhanden sind. Diese Bedingung kann wie folgt formalisiert werden: ($H_{vorne,nah} = false \wedge Ziel_{nah} = false$), wobei H für Hindernis und Z für Ziel steht. Die Abbruchbedingung des Basis-Knotens (abstrakte Aktion "zum Ziel fahren") stellt zuerst die Negation der Vorbedingung dar. Die Abbruchbedingung der Basis-Knoten "zum Ziel fahren" ist erfüllt, wenn das Ziel erreicht ist, oder sich ein Hindernis vor dem Agenten befindet ($Ziel_{nah} = true \vee H_{vorne,nah} = true$). Wenn eine der Abbruchbedingungen erfüllt ist, und der Basis-Knoten im Status "Intention" ist, wird die abstrakte Aktion "zum Ziel fahren" in den Status "abgebrochen" versetzt. Wenn die Geduld des Agenten überschritten wird, wird der Status des Basis-Knotens "zum Ziel fahren" in den Status "abgebrochen" versetzt. Dabei wird diese abstrakte Aktion nicht aus der Menge der verfügbaren Aktionen ausgeschlossen. Die Geduld wird mithilfe der vom Agenten zurückgelegten Distanz simuliert. Wenn die zurückgelegte Distanz einen maximalen, vom Nutzer vorgegebenen Wert überschreitet, wird davon ausgegangen, dass der Agent eine Aktion zu lange ausführt, und es wird nach möglichen Alternativen gesucht.

7.3.4 Abstrakte Aktion "Hindernis rechts/links umfahren"

Wenn die abstrakte Aktion "Hindernis rechts/links umfahren" ausgewählt ist, soll sich der Agent in eine vorgegebene Richtung mit vorgegebenem Abstand am Hindernis entlang bewegen.
In Kapitel 6 sind die möglichen Methoden[42] zum Erzielen der reaktiven Fähigkeiten "Hindernis rechts / links umfahren" beschrieben. Aus diesen reaktiven Fähigkeiten werden die vom DPA-RL-Agenten benötigten abstrakten Aktionen "Hindernis rechts/links umfahren" gebildet. Die abstrakten Aktionen sind die Kindknoten der RL-Choice-Knoten. In Abhängigkeit davon, ob die abstrakte Aktion aus einer Kontrollfunktion besteht (im Fall mit erzielten kNN) oder aus mehreren Kontrollfunktionen zusammengesetzt (im Fall mit Regelungstechnik) wird, wird die abstrakte Aktion als ein Basis-Knoten oder als ein Choice-Knoten realisiert. Diese beiden Möglichkeiten werden in Hinblick auf die Bestimmung der Vor- und Abbruchbedingungen der Basis- bzw. Choice-Knoten unten genauer betrachtet.

[42] Anwendung des evolutionären Ansatzes oder der klassischen Regelungsmethoden

7.3.4.1 "Hindernis rechts/links umfahren" als Basis-Knoten (kNN)

Die erste Möglichkeit die gewünschten abstrakten Aktionen zu erzielen wird hier mithilfe künstlicher neuronaler Netze (kNN) realisiert. Als Antwort auf die lokalen Informationen, die die IR-Sensoren liefern, werden die Befehle auf die Motoren des Scorpion-Roboters mithilfe der verwendeten kNN umgerechnet. Die für das Training der neuronalen Netze verwendeten Szenarien, die Fitnessfunktion und die verwendeten neuronalen Netze wurden in Kapitel 6.2 genauer erläutert. Die beiden verwendeten kNN, siehe Netz zur Aktion "Hindernis rechts umfahren" bzw. "Hindernis links umfahren" in Abbildung 6.5, repräsentieren die benötigten Kontrollfunktionen für die Bildung der beiden abstrakten Aktionen. Die abstrakten Aktionen "Hindernis rechts/links umfahren" werden im Optionenbaum als zwei Basis-Knoten realisiert.
Die Vorbedingung für den Basis-Knoten "Hindernis rechts umfahren" ist dann erfüllt, wenn ein Hindernis spürbar ist ($H_{vorne,nah} = true \vee H_{vorne,mittel} = true \vee H_{vorne,weit} = true \vee H_{links,nah} = true \vee H_{links,mittel} = true \vee H_{links,weit} = true$). Wenn die Vorbedingung der Kindknoten der RL-Choice-Knoten erfüllt ist, dann kann diese abstrakte Aktion zur ausführbaren Aktionsmenge hinzugefügt werden.
Ebenso ist die Vorbedingung für den Basis-Knoten "Hindernis links umfahren" erfüllt, wenn ein Hindernis spürbar ist ($H_{vorne,nah} = true \vee H_{vorne,mittel} = true \vee H_{vorne,weit} = true \vee H_{rechts,nah} = true \vee H_{rechts,mittel} = true \vee H_{rechts,weit} = true$).
Die Abbruchbedingung für beide abstrakten Aktionen ist nur erfüllt, wenn die vom Agenten zurückgelegte Distanz d überschritten wird, oder kein Hindernis vom Roboter wahrgenommen wird.

7.3.4.2 "Hindernis rechts/links umfahren" als Choice-Knoten (Regelungstechnik)

Eine weitere Möglichkeit, die abstrakten Aktionen "Hindernis links umfahren" und "Hindernis rechts umfahren" zu realisieren, ist die Anwendung der Regelungstechnik. In diesem Fall besteht eine abstrakte Aktion aus zwei zusammengesetzten Kontrollfunktionen. Die mit den Methoden der Regelungstechnik realisierte abstrakte Aktion ist komplexer als die Realisierung mithilfe künstlicher neuronaler Netze.
Die Choice-Optionen "Hindernis links umfahren" und "Hindernis rechts umfahren" navigieren den Roboter um ein Hindernis herum, indem die Blätter (Basis-Knoten) "Drehe rechts/links" und "Hindernis rechts/links entlang fahren" abwechselnd aktiviert werden. Die Schleife mit beiden Kontrollfunktionen ist in Abbildung 6.8 dargestellt. Zwei Kontrollfunktionen werden als Basis-Knoten im Optionenbaum repräsentiert. Diese Basis-Knoten werden über einen Choice-Knoten mit einem exklusiven "Oder" miteinander verbunden, so dass entweder die eine oder die andere Kontrollfunktion vom Choice-Knoten ausgeführt wird. Der Choice-Knoten ist dann der Kindknoten des RL-Choice-Knotens und bildet die gewünschte abstrakte Aktion "Hindernis rechts/links umfahren". Diese aufgestellten Vorbedingungen für die Realisierung der abstrakten Aktion "Hindernis rechts/links umfahren" mit der Regelungstechnik und Choice-Knoten sind den aufgestellten Vorbedingungen für die Realisierung mit neuronalen Netzen und Basis-Knoten gleich.
Der Optionenbaum wird maximal drei Ebenen beinhalten: Die erste Ebene, auf der der RL-Choice-Knoten liegt, die zweite Ebene, auf der die abstrakten Aktionen liegen und die dritte Ebene, auf der die ggf. vorhandenen Teilschritte, aus denen die abstrakten Aktionen zusammengestellt werden, liegen.
Die Vorbedingung der Choice-Option "Hindernis rechts umfahren" ist wahr, wenn entweder ein Hindernis vorne oder auf der linken Seite der Abstandssensoren gespürt wird.
Also ($H_{vorne,nah} = true \vee H_{vorne,mittel} = true \vee H_{vorne,weit} = true \vee H_{links,nah} = true \vee$

$H_{links,mittel} = true \vee H_{links,weit} = true$).
Ebenso ist die Vorbedingung der Choice-Option "Hindernis links umfahren" erfüllt, wenn entweder ein Hindernis vorne oder auf der rechten Seite der Abstandssensoren gespürt wird, also ($H_{vorne,nah} = true \vee H_{vorne,mittel} = true \vee H_{vorne,weit} = true \vee H_{rechts,nah} = true \vee H_{rechts,mittel} = true \vee H_{rechts,weit} = true$).
Bei der Abbruchbedingung der Choice-Knoten "Hindernis links umfahren" wird zunächst überprüft, ob sich ein Hindernis auf der vorderen oder der linken Seite des Roboters befindet. Ist das nicht der Fall, wird noch zusätzlich geprüft, ob die Differenz zwischen der aktuellen Orientierung des Roboters und der Orientierung des Roboters zum Zeitpunkt, als das letzte Mal keine Hindernisse wahrgenommen worden sind, mehr als $180°$ beträgt. Ist das der Fall, so wird die Ausführung der Aktion abgebrochen. Dies ermöglicht eine Umrundung von Hindernissen, die so abrupt enden, dass der Roboter zwischenzeitlich den Kontakt zum Hindernis verliert. [43] Eine weitere Abbruchbedingung der Ausführung ist die Überschreitung der vom Nutzer vorgegebenen maximal zurückzulegenden Distanz d seit der letzten Aktivierung der Option.
Die Abbruchbedingungen des Choice-Knotens "Hindernis rechts umfahren" sind symmetrisch zu den Abbruchbedingungen des Choice-Knotens "Hindernis links umfahren" definiert. Zur Bestimmung der Abbruchbedingungen werden dieselben Berechnungen bezüglich der rechten Seite des Roboters durchgeführt.
Die Kontrollfunktion "Drehe rechts/links" wird als ein Basis-Knoten im Optionenbaum repräsentiert. Diese Kontrollfunktion dreht den Agenten solange nach links bzw. rechts vom Hindernis weg, bis eine Weiterfahrt nach vorne wieder möglich ist, siehe auch Kapitel 6.3. Die Vorbedingung der Basis-Knoten "Drehe rechts/links", wird erfüllt, wenn sich nahe an der Vorderseite des Roboters ein Hindernis befindet ($H_{vorne,nah} = true$).
Die Abbruchbedingung der Basis-Knoten "Drehe rechts/links" stellt die Negation der Vorbedingung dar. Die Kontrollfunktionen "Hindernis rechts/links entlang fahren" steuern den Agenten am Hindernis so entlang, dass der Agent den vorgegebenen Abstand zum Hindernis beibehält. Die Vorbedingung der Basis-Knoten "Hindernis rechts/links entlang fahren" wird erfüllt, wenn sich auf der Vorderseite des Roboters kein Hindernis befindet und das Hindernis auf der rechten/linken Seite spürbar ist, also

$$(H_{vorne,nah} = false) \wedge (H_{rechts,\{nah,mittel,weit\}} = true \vee H_{links,\{nah,mittel,weit\}} = true).$$

Die Abbruchbedingungen der Basis-Knoten "Hindernis rechts/links entlang fahren" sind Negationen der Vorbedingungen.

7.4 Dynamischer Aufbau des Aktions- und Zustandsraums

Der DPA-RL-Agent ist eine Struktur mit dynamisch ausgebauten Zustands- und Aktionsräumen. Die Zustands- und Aktionsräume werden nach Bedarf ausgebaut, so dass nur die wichtigen Entscheidungsknoten und eine dadurch spärliche Besetzung des Zustandsraumes und wichtige Zwischenziele den Lernprozess beeinflussen. Diese Tatsache trägt entscheidend zur Geschwindigkeit des Lernprozesses im entwickelten DPA-RL-Agenten bei.

[43] Für die Realisierung der abstrakten Aktion mit kNN ist diese komplexe Abbruchbedingung nicht nötig. In der Offlinephase, beim Training der neuronalen Netze werden Szenarien mit solchen abrupten Änderungen des Hindernisses aufgestellt. Die Netze werden trainiert mit solchen Schwierigkeiten umzugehen.

7.4.1 Dynamischer Aufbau des Zustandsraums

Die wichtigen Stellen, an denen die Ausführung der abstrakten Aktion unterbrochen wurde, und eine Entscheidung über die neue abstrakte Aktion getroffen werden muss, werden als interne Zustände im Zustandsraum der RL-Choice-Knoten erfasst. Wenn diese Stellen im Raum vom DPA-RL-Agenten erneut angefahren werden, wird der Agent auf die gemachten Erfahrungen zurückgreifen.
Die dynamisch aufgebaute Struktur des Zustandsraumes wird als eine Liste Z gespeichert. Die Beiträge dieser Liste beinhalten den internen Zustand des Agenten. Er besteht aus der Position des Agenten und dessen Orientierung, die aus Odometriedaten entnommen werden. In der vorliegenden Arbeit werden diese Daten im Koordinatensystem mit dem Ursprung in der Startposition und der Orientierung des Agenten aus den Odometriedaten des Scorpion-Roboters entnommen. Die Einträge in der Liste werden über diesen internen Zustand verwaltet. Die lokalen Informationen aus den IR-Sensoren werden auch für jeden der neuen Einträge mitgespeichert.
Im Algorithmus 7.1 ist der dynamische Aufbau des Zustandsraumes Z dargestellt. Als Input für den Algorithmus dient die aktuelle Position und Orientierung des Agenten $s_t := ((x_t, y_t), o_t)$ sowie die lokale Modellumgebung in diesem Zustand $M_{((x_t,y_t),o_t)}$. Als Output wird der aktuelle Zustandsraum Z und der Index für die aktuelle Position aus dem Zustandsraum Z ausgegeben. Wenn ein dem aktuellen Zustand s_t ähnlicher Zustand in der Menge der Zustände Z existiert, wird der Zustandsraum nicht erweitert, sondern nur der Index des ähnlichen Zustandes aus der Liste Z zurückgegeben. Wenn der aktuelle Zustand nicht in der Menge der Zustände vertreten ist, wird die Liste Z um den aktuellen Zustand erweitert und der Index des letzten Elements zurückgegeben. Die Merkmalmenge F_{s_t}, siehe Kapitel 3.4.3, wird aus dem zum Index des aktuellen Zustands korrespondierenden Zustand zusammengesetzt.
Der Algorithmus beinhaltet zwei Parameter. Mithilfe dieser Parameter wird die Ähnlichkeit des neu-

Algorithmus 7.1 Algorithmus zum dynamischen Aufbau des Zustandsraumes

$(x_t, y_t)^T, o_t$ ▷ Die aktuelle Position und Orientierung des Agenten
for all $i = 1, \ldots |Z|$ **do** ▷ $|Z|$ ist die Anzahl der Knoten im $Graph'$
  if $||(x_t, y_t)^T - (x^i, y^i)^T|| < SW_d \wedge |o_t - o_i| < SW_o$ **then**
    Return (Z und i)
$Z := Anfügen(Z, \{((x_t, y_t), o_t), M_{((x_t,y_t),o_t)}\})$
Return (Z und $|Z|$)

en Zustandes s_t mit den existierenden Zuständen im Zustandsraum Z gemessen. Diese Parameter sind $SW_d := 50\ cm$ für die Bestimmung der Ähnlichkeit der aktuellen Position mit den vorhandenen Positionen der Zustände im Zustandsraum Z. Der Parameter $SW_o = 45°$ ist für den Abgleich der aktuellen Orientierung mit den existierenden Orientierungen der Zustände im Zustandsraum bestimmt. Die optimalen Werte für den Parameter SW_d und SW_o sind in der Arbeit [Mai, 2010] bestimmt worden. Da der ganze Bereich mit einem Durchmesser von $50\ cm$ einen Zustand repräsentiert, ähnelt dieser Algorithmus der Coarse-Coding-Methode. Der Parametervektor wird zusammen mit dem Zustandsraum automatisch erweitert, so dass die Anzahl der Zustände im Zustandsraum Z immer mit der Anzahl der Parameter im Parametervektor θ_a übereinstimmt.

7.4.2 Dynamischer Aufbau der Aktionsmenge

Drei grundlegende, abstrakte Aktionen erlauben es dem Agenten nicht immer den optimalen Weg zu erlernen, da die Lösung vom Aufbau der Umgebung abhängt. Die Notwendigkeit der Anwendung von

Zwischenzielen im Lernprozess für ein Semi-Markow-Entscheidungsproblem wurde z. B. schon in [Sutton et al., 1999] gezeigt.
Der gestrichelte Pfeil in Abbildung 7.8 deutet den optimalen Weg an, der aber mit nur drei vorgegebenen, abstrakten Aktionen nicht erzielt werden kann. Der in der Abbildung durch einen durchgezogenen Pfeil angedeutete Weg ist bei der Anwendung von drei vorgegebenen abstrakten Aktionen entstanden. Der Agent wählt zuerst die Aktion ”zum Ziel fahren”. Die nächste benötigte abstrakte Aktion ist ”Hindernis rechts umfahren”. Diese Aktion wird solange ausgewählt, bis das Hindernis den Agenten nicht mehr daran stört zum Ziel zu fahren, siehe Weg s_1, s_2, s_3 in der Abbildung. Anschließend wird die Aktion ”zum Ziel fahren” im Zustand s_3 ausgewählt. Dieser Weg ist nicht optimal. Wenn der Agent als erste Entscheidung ”fahre zum Wendepunkt” (den Zustand s_3) wählte, wäre der Weg zum Ziel kürzer, vergleiche den gestrichelten Pfeil in der Abbildung. Solche wichtigen Punkte können, müssen aber nicht existieren. Der Wendepunkt ist in diesem Beispiel ein Zwischenziel auf dem optimalen Weg zum Ziel. Die Existenz der Zwischenziele hängt von der Topologie der Umgebung, in der sich der Agent befindet, ab.
Die Aktionsmenge des Agenten kann dynamisch um die möglichen Zwischenziele erweitert werden, die durch solche wichtigen Punkte wie der Wendepunkt aus dem Beispiel in Abbildung 7.8 repräsentiert werden.
Der angewendete Ansatz zur Bestimmung der Zwischenziele wird im vorliegenden Unterkapitel eingeführt. Die neuen, abstrakten Aktionen können durch Vorgabe der automatisch gewonnenen Zwischenziele ($ZZ_i, i = 1, \ldots, P$) in der reaktiven Fähigkeit ”zum Ziel fahren” gebildet werden. Die neu gebildete abstrakte Aktion ”fahre zu ZZ_i” wird als ein neuer Kindknoten der RL-Choice-Option hinzugefügt. Durch diesen Schritt wird auch die Aktionsmenge, die in der RL-Choice-Option angewendet wird, erweitert. Das angesprochene Zwischenziel liegt an wichtigen Wendepunkten. Die Bestimmung der neuen

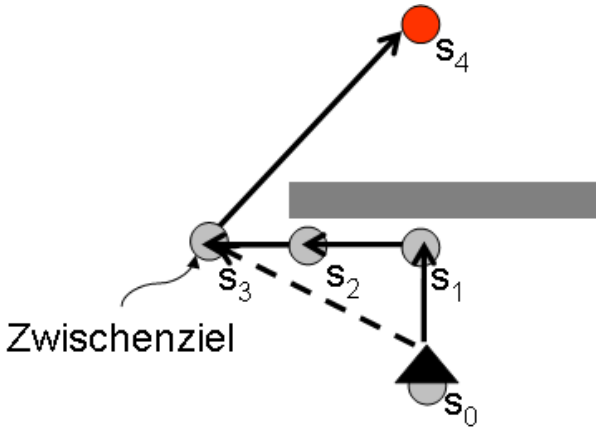


Abbildung 7.8: Abhängigkeit der erlernten Lösung von der Hindernisform

abstrakten Aktionen wird erst durchgeführt, wenn die erste Episode vom Agenten durchlaufen ist. Eine Episode entspricht einem Durchlauf vom Start zum Ziel, dabei wird ein Teil des Zustandsraumes aufgebaut. Der Graph beinhaltet in jedem Fall einen Weg vom Start des Agenten bis zum Ziel.
Der $Graph'$ wird zur Bestimmung des kürzesten Weges zum Zielknoten verwendet. Er wird mit dem Projektionsschritt des Zustandsraums Z im Teilgraphen $Graph'$ bestimmt, siehe Unterkapitel 7.3.2. Der kürzeste Weg wird mit dem A^*-Algorithmus [Hart et al., 1968] bestimmt. Der aus dem $Graph'$ bestimmte kürzeste $Weg = \{k_1, \ldots, k_N\}$ wird zur Bestimmung der Zwischenziele verwendet. Der Weg ist eine geordnete Reihe von Knoten im Raum von der Startposition zur Zielposition des Agenten, ein Knoten wird durch die Position im Raum repräsentiert. Die Orientierung des Agenten, in der der Agent die Kno-

ten angefahren hat, hat für die Bestimmung der Zwischenziele keine Bedeutung.
Die grobe Idee besteht darin, dass die Teilwege gefunden werden sollen, die Umwege im Raum darstellen. Umwege können wie im oben genannten Beispiel entstehen, zum Beispiel durch eine Fahrt um das Hindernis herum. In Abbildung 7.8 stellt der Weg (s_0, s_1, s_2, s_3) einen solchen Umweg dar. Wenn der Agent direkt zum Wendepunkt fährt, also den Weg (s_0, s_3), wird die gewünschte Abkürzung erreicht.
Ein potenzieller Umweg kann mithilfe von zwei Werten abgeschätzt werden, durch die Länge $Länge$ der möglichen Umwege und den Winkel α_δ der zurückgelegten Bögen. Zwei Schwellenwerte werden zur Bestimmung der Zwischenziele benötigt. Der erste Schwellenwert $Länge_\delta$ ist eine Art gesparte Weglänge. Der zweite Schwellenwert α_δ ist der Schwellenwert für den unnötig gemachten Bogen. Mit dem angewendeten Ansatz wird nach möglichen Umwegen im kürzesten $Weg = \{k_1, \ldots, k_N\}$ gesucht. Der Weg wird nach den Teilstrecken aufgeteilt $Weg = \{\{k_1, \ldots, k_{T_1}\}, \{k_{T_1+1}, \ldots, k_{T_2}\}, \ldots \{k_{T_n+1}, \ldots k_N\}\}$, wobei jeder Teilweg $\{k_{T_j+1}, \ldots, k_{T_{j+1}}\}$ über den Winkelverlauf α_δ in dieser Teilstrecke bestimmt wird. Mithilfe des Winkels α_δ wird der zurückgelegte Bogen, also Umweg, abgeschätzt. Der Winkel wird zwischen zwei Vektoren berechnet, der erste Vektor ist ein vom Startknoten der Teilstrecke zur Zielposition gerichteter, normierter Vektor. Der zweite Vektor ist ein vom aktuellen Knoten k_t zum Zielpunkt $t = 2, \ldots, N$ gerichteter, normierter Vektor. In dem Fall, dass der Winkel $(\alpha_\delta)_{t-1} \leq (\alpha_\delta)_t$ im Übergang von einem Knoten zum nächsten wächst, wird eine Teilstrecke gebildet. Wenn der Winkel nicht mehr wächst $\alpha_{\delta t} < \alpha_{\delta t-1}$ wird die Bildung der aktuellen Teilstrecke beendet und eine neue Teilstrecke, mit dem Startknoten im nächsten Knoten von Weg angefangen. Der Ansatz wird über alle Knoten im Weg durchgeführt, so dass am Ende der Weg in Teilwege zerlegt ist.
In Abbildung 7.9 wird der Ansatz, mit dessen Hilfe der gesuchte Umweg erkannt werden kann, gra-

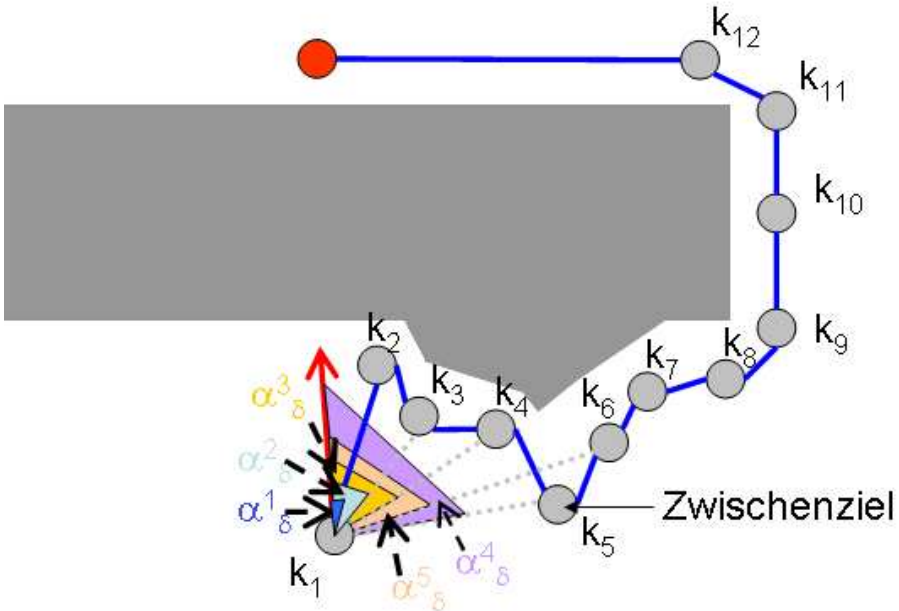


Abbildung 7.9: Graphische Darstellung des Ablaufs des angewendeten Ansatzes. Der rote Pfeil ist der Richtungsvektor vom ersten Knoten des ersten Teilwegs zum Ziel.

phisch dargestellt. Die unterschiedlichen Winkel sind durch unterschiedliche Farben gekennzeichnet. In dem Fall, dass der Winkel α_δ von einem Schritt zum nächsten wächst, wird eine Teilstrecke gebildet. Diese Teilstrecke ist der vermutliche Umweg. Im Beispiel ist $\{k_1, k_2, k_3, k_4, k_5\}$ eine Teilstrecke, da $\alpha_\delta^1 < \alpha_\delta^2 < \alpha_\delta^3 < \alpha_\delta^4$, siehe Abbildung. Wenn dann $\alpha_\delta^t < \alpha_\delta^{t-1}$, siehe $\alpha_\delta^5 < \alpha_\delta^4$ in der Abbildung,

dann wird geprüft, ob die gebildete Strecke den vordefinierten Nutzen[44] bringt. Diese Bedingungen für den vordefinierten Nutzen beinhalten die beiden oben eingeführten Schwellenwerte ($\delta_\alpha, Länge_\delta$) für die gesparte Länge der Teilstrecke $Länge$ und den Winkel des gemachten Bogens α_δ. Wenn dieser Winkel klein ist, nimmt der Agent einen relativ direkten Weg zum Ziel. In diesem Fall ist keine Abkürzung notwendig. Wenn der Winkel $\alpha_\delta > \delta_\alpha$ und die gesparte Länge $Länge > Länge_\delta$ groß genug sind, wird die gebildete Teilstrecke als ein Umweg angesehen. Der letzte Knoten dieser Teilstrecke wird zur Menge der Zwischenziele hinzugefügt.
Die Anzahl der Zwischenziele kann so groß wie nötig gewählt werden. Nur bei der Konstruktion des Optionenbaums muss diese zukünftige Menge von Zwischenzielen berücksichtigt werden. Je größer die Anzahl der Zwischenziele ist, desto langsamer wird der Lernvorgang sein, da die Anzahl der zu untersuchenden Zustand-Aktions-Paare potenziell mit der Anzahl der möglichen Aktionen wächst. Aus diesem Grund wird die Grenze für die maximale Anzahl der möglichen Zwischenziele in der vorliegenden Arbeit auf einen relativ kleinen Wert von 5 gesetzt.
Nach jeder Episode kann ein neuer kürzester Weg bestimmt werden. Auf dem neuen mit dem A^*-Algorithmus bestimmten Weg können weitere Zwischenziele einen Umweg ersparen. Die neuen Kandidaten der Zwischenziele werden mithilfe der Maße, die den gesparten Weg ($Länge$) abschätzen, mit schon existierenden Zwischenzielen verglichen.

7.5 Perspektiven des DPA-RL-Agenten

Eine Möglichkeit das dargestellte Konzept des DPA-RL-Agenten zu erweitern liegt in der Verwendung eines komplexeren Optionenbaums.
Die erste Möglichkeit wäre statt einen mehrere RL-Choice-Knoten auf einer Ebene des Optionenbaums aufzustellen und diese RL-Choice-Knoten mit übergeordneten Sequenzknoten zu steuern. Die Verwaltung der RL-Knoten durch die übergeordneten Sequenzknoten kann die Inspektionsaufgabe erfolgreich vollziehen. Die RL-Knoten würden in diesem Fall den Weg von einem zu inspizierenden Objekt bis zum nächsten beinhalten. Der Sequenzknoten würde die Reihenfolge der Aktivierung der im RL-Choice-Knoten erlernten Weg von einem Inspektionsobjekt zum anderen vorgeben.
Eine weitere Möglichkeit das dargestellte Konzept des DPA-RL-Agenten zu erweitern liegt in der Anwendung eines Optionenbaums, der aus zwei oder sogar mehr Ebenen von RL-Choice-Knoten besteht. Je nach Mission des Agenten können neue gestellte Navigationsprobleme bereits erlernte optimale Teilstrecken beinhalten. Wenn aber bereits erlernte Teilstrecken mitbenutzt werden sollen, um das globale Problem zu lösen, wird die Aufgabe deutlich komplexer. In diesem Fall repräsentieren die erlernten RL-Choice-Knoten Teilstrecken, so dass die Lösung für ein Navigationsproblem aus den existierenden, erlernten Lösungen zusammengesetzt werden kann. Für übergeordnete RL-Choice-Knoten dienen die erlernten Strecken als Aktionen. Da die Äste eines RL-Choice-Knotens der DPA-RL-Architektur dynamisch ausbaubar sind, können die unbekannten Teilstrecken ohne die Notwendigkeit der manuellen Veränderung der Baumstruktur eingeführt werden.
Ein Optionenbaum mit mehr als einer Schicht der RL-Choice-Knoten ist eine viel flexiblere Struktur. Für die Anwendung so einer Struktur müssen aber viele Probleme gelöst werden. In den vorliegenden Überlegungen, wie die Forschung weiter gehen kann, werden die Methoden genannt, die die erkannten Probleme lösen.
Ein Beispiel für eine solche Art der Erweiterung kann wie folgt aussehen. Es wird angenommen, dass eine Strategie existiert, die den Agenten auf dem optimalen Weg vom Start A zum Ziel B führt. Au-

[44] Also sich die vermutete Abkürzung als signifikant auszeichnet.

ßerdem wird angenommen, dass eine weitere Strategie existiert, die den Agenten schnell vom Punkt C nach Punkt D steuert. Die neue Aufgabe lautet, dass der Agent von Punkt A nach Punkt D fahren muss. Wobei der Agent während der Ausführung der Mission feststellt, dass der optimale Weg für die Strecke von A nach D die schon erlernten Teilstrecken (A $\rightarrow$ B und C $\rightarrow$ D) beinhaltet. Zwei Teilstrecken sind in zwei RL-Choice-Knoten vorhanden, und der Agent muss nur die Strecke B $\rightarrow$ C erlernen, um die Aufgabe zu lösen. Die getroffene Annahme ist von großer Bedeutung, und deren Schwierigkeit darf nicht unterschätzt werden. Die Schwierigkeit liegt darin, dass für jede der erlernten Teilstrecken eigene Koordinatensysteme vorliegen. Dadurch wird der Agent mit einer Art Matching-Problem konfrontiert. Dieses Problem kann auf unterschiedliche Art und Weise gelöst werden. Die erste Möglichkeit besteht in der Verwaltung und dem Aufbau einer globalen Karte, so dass die erlernten Teile zu einem globalen Koordinatensystem zusammengeführt werden. In diesem Fall müssen auch die neu erlernten Teilstrecken in das globale Koordinatensystem gebracht oder "gematcht" werden. Die zweite Möglichkeit die erlernten Teilstrecken zu berücksichtigen kann durch Bestimmung und Anwendung ähnlicher, schon erlernter Situationen erfolgen. In diesem Fall können Matching-Methoden zum Beispiel die Partikelfilter-Methode [Thrun et al., 2005], [Thrun et al., 2000] angewendet werden.
Mit dem Partikelfilteransatz kann eine schon erlernte Strecke bzw. Teilstrecke[45], die ein ähnliches Ziel bzw. Zwischenziel und eine ähnliche Umgebung hat, gefunden werden. Da jede der verwendeten Strecken eine Erkundungsphase benötigt, werden die vielen Aktionen ausprobiert und die für den Partikelfilteransatz benötigten Schritte gemacht. Die benötigten lokalen Umgebungsmodelle für die bekannten Strecken sind vorhanden. Durch diese Art der Erweiterung werden die gesammelten Erfahrungen im Lernprozess mitberücksichtigt. Das Problem liegt darin, dass sehr viele Teilstrecken theoretisch verfolgt werden müssen. Der spärlich aufgebaute Zustandsraum bereitet einen großen Vorteil. Die Anzahl der Zustände pro Szenario ist nicht groß und die Bestimmung des aktuellen Zustandes in der vorhandenen Aufgabe ist möglich.

[45] Eine Teilstrecke ist eine Strecke zwischen dem Start und dem Ziel oder eine Strecke zwischen einem Zwischenziel und einem anderen Zwischenziel.

Kapitel 8

Ergebnisse für das Navigationsproblem mit dem DPA-RL-Agenten

Der in Kapitel 7 vorgestellte DPA-RL-Agent stellt die endgültige Lösung für das in der vorliegenden Arbeit betrachtete Navigationsproblem dar. In diesem Kapitel werden die entwickelten Agenten hinsichtlich Effizienz ihres Lernprozesses und Güte der Lösung für das verwendete Navigationsproblem untersucht. In Kapitel 8.1 werden zuerst die verfolgten Ziele formuliert. Die Randbedingungen, die angewendete Parametrisierung und der Versuchsaufbau werden in diesem Kapitel erläutert.
In erster Linie ist die Fähigkeit des Agenten zum Lernen und die Effizienz des Lernprozesses von Bedeutung. Die Lernfähigkeit des DPA-RL-Agenten wird in Kapitel 8.2 untersucht.
Die Auswirkung der in Kapitel 6 eingeführten unterschiedlichen Realisierungen der reaktiven Fähigkeiten auf den Lernprozess, die für den Aufbau der abstrakten Aktionen verwendet werden, ist von großem Interesse. Realisierungen der abstrakten Aktion ”Hindernis rechts/links umfahren” werden in Kapitel 8.3 miteinander verglichen.
In Kapitel 8.4 wird die Auswirkung der in Kapitel 4 präsentierten heuristischen Erweiterung des Lernprozesses für den DPA-RL-Agenten untersucht.
Eine kurze Zusammenfassung schließt diesen Teil ab, siehe Kapitel 8.5.
Der DPA-RL-Agent wird für die Lösung des Navigationsproblems unter realen Bedingungen eingesetzt. Die Ergebnisse sind u.a. als Videomaterial dokumentiert.

8.1 Zielsetzung und Versuchsaufbau

Einen Gesamtüberblick über die wichtigsten Versuchsreihen gibt die folgende Auflistung:

- Analyse der Lernfähigkeit hinsichtlich Adaptionsfähigkeit, Generalisierbarkeit, Reaktivität
- Unterschiedliche Realisierungen der reaktiven Fähigkeiten: PID-Regler, NEAT
- Untersuchung des heuristischen Einflusses im Belohnungsmodell des DPA-RL-Agenten

Die Unterschiede der untersuchten Konfigurationen des DPA-RL-Agenten sind nicht immer direkt erfassbar, so dass viele Versuche teilweise in der Simulation durchgeführt werden, um statistisch aussagekräftige Ergebnisse und Vergleiche zu bestimmen. Die wichtigsten Fähigkeiten des entwickelten Systems, wie Lernfähigkeit und Sensibilität gegenüber den von Sensoren erzeugten Ungenauigkeiten, werden in der Realität untersucht.
Jeder in diesem Kapitel beschriebene Versuch ist, wenn keine weiteren Bemerkungen zum Versuchsaufbau gemacht werden, gleich aufgebaut. Für jeden der durchgeführten Versuche ist am Anfang kein Wissen über das Umgebungsmodell vorhanden. Der Agent erhält nur die Informationen über seinen aktuellen Zustand und die Position des Ziels in Bezug auf seinen aktuellen Zustand. Das Ziel ist als Punkt im

Raum im globalen Umgebungskoordinatensystem vorgegeben. Der Ursprung des Koordinatensystems des Agenten liegt in seiner Startposition. Nur wenige Episoden werden benötigt[46], um einen guten Weg vom Start zum Ziel zu finden. Mit jeder Episode verbessert sich das Verhalten des Agenten. Je nach den gestellten Zielen werden entweder gleiche Versuche mehrere Male in der Simulation durchgeführt oder der Aufbau des Szenarios geändert, um die untersuchte Fähigkeit des Systems auf die Probe zu stellen.
Die Versuche sind sowohl in der Realität, als auch in der realitätsnahen Simulation durchgeführt worden. Der Ablauf jedes Versuchs ist gleich und beinhaltet den folgenden Parametersatz: $\alpha = 0.5, \epsilon = 0.05, \gamma = 0.8, \lambda = 0.8, d = 50$. Die Konstante d steht für die maximale vom Nutzer vorgegebene zurückgelegte Distanz. Die zur Diskretisierung und zum Aufbau des Zustandsraums benötigten Parameter, vergleiche Algorithmus 7.1, werden wie folgt gesetzt: $SW_d = 50\ cm$, $SW_o = 45°$. Jedes Mal wird der Mechanismus zur automatischen Zwischenzielbestimmung eingesetzt.
Die unterschiedlichen Realisierungen und Erweiterungen der angewendeten Methoden sollen miteinander verglichen werden, um Aussagen über die beste Konfiguration für den DPA-RL-Agenten treffen zu können. Das grundlegende Vergleichskriterium ist der Lernfortschritt (engl.: learning progress, abgekürzt LP genannt), siehe Kapitel 3.6. Beim Vergleich der unterschiedlichen Realisierungen der abstrakten Aktion ist der Lernfortschritt, der auch die Lerngeschwindigkeit repräsentiert, für jede Modifikation ungefähr gleich. Für diesen Vergleich wird ein Kriterium eingeführt, das die Glattheit der vom Agenten gefahrenen Trajektorien misst. Dieses Kriterium wurde auch in Kapitel 6.4 verwendet, siehe Gleichung (6.5).

Bemerkung 8.1.1 (DPA-RL-Agent und Aufbau der Umgebung). *Der Lernfortschritt (Lerngeschwindigkeit) des DPA-RL-Agenten ist im Gegensatz zu vielen klassischen auf MDP basierenden RL-Ansätzen zur Roboter-Navigation nicht so stark von der Größe des Areals der Umgebung abhängig, sondern vielmehr von ihrer Beschaffenheit. Der Grund für diese Tatsache liegt darin, dass nur wichtige Entscheidungsknoten an den erlernten Entscheidungsketten teilnehmen, die Zustands- und Aktionsräume werden dynamisch und abhängig vom Bedarf ausgebaut.*

Diese Bemerkung ist wichtig für die Motivation zum Aufbau der Szenarien. Für die Verdeutlichung der in der Bemerkung gemachten Aussage wird an dieser Stelle ein Beispiel eingeführt. Es ist mit dem Unterschied zwischen einer Fahrt auf der Autobahn und in einer verkehrsberuhigten Zone vergleichbar. Bei der Fahrt auf der Autobahn soll der Fahrer auf einer langen Strecke dem Straßenverlauf einfach nur folgen und nur selten Entscheidungen über Autobahn- bzw. Fahrstreifenwechsel treffen. In einer verkehrsberuhigten Zone hingegen muss er auf einer kurzen Strecke permanent zwischen den einzelnen Wegen entscheiden. Die Länge der Route sagt nichts über die Komplexität der Aufgabe aus.
Im Folgenden werden Bildfolgen dargestellt, die automatisch aus dem Videomaterial erzeugt werden. Eine Bildfolge wird mithilfe des Vorschaumodus der Software SMPlayer automatisch aus der Videosequenz erzeugt. Das Zeitintervall ist zwischen jedem dargestellten Bild in der Bildfolge gleich. Jedes Bild der Bildfolge ist mit einem Zeitstempel signiert. Der Zeitstempel ist in der rechten unteren Ecke jedes Bildes abgebildet. In einigen Bildfolgen liegen zwischen Ende und Anfang von zwei Episoden ein oder sogar mehrere Bilder, da die Kamera kontinuierlich durchgelaufen ist, und zwischen zwei Episoden die Zeit benötigt wurde, um den Roboter-Agenten vom Ziel- zum Startpunkt zu transportieren, vergleiche z.B. das Bild in Abbildung 8.2 zwischen "Episode 3, Ende" und "Episode 4, Start". Das Ziel ist mit einem grünen Rechteck in der dargestellten Bildfolge gekennzeichnet. Der Startpunkt wird als gelbes Rechteck in der Bildsequenz angedeutet. Die Episoden sind in jeder dargestellten Bildfolge abgegrenzt, so dass

[46] In der Regel sind schon 3 bis 4 Episoden ausreichend, um einen guten Weg zu finden.

der Start, die Nummer und der Endzustand der aktuellen Episode durch ”Episode N, Start” und ”Episode N, Ende” gekennzeichnet sind. Dazwischen liegt die Teilbildfolge, die zur gekennzeichneten Episode gehört. Eine Episode startet immer vom gleichen Punkt aus, und das Ziel des Agenten bleibt für jede Episode eines Szenarios gleich. Die Umgebung in einem Szenario kann je nach Untersuchungszweck verändert werden.

8.2 Lernfähigkeit des DPA-RL-Agenten

Bevor die Lernfähigkeit des DPA-RL-Agenten geprüft wird, wird zuerst der Begriff Lernfähigkeit genauer erläutert. Die Möglichkeit sich passend zur Situation zu adaptieren und das gewonnene Wissen in ähnlichen Situationen anwenden zu können, also zu verallgemeinern bzw. generalisieren, sind die wichtigsten Merkmale, die der entwickelte lernbasierte Agent nachweisen soll. Somit wird die Lernfähigkeit des Agenten geprüft, indem die erwarteten Eigenschaften die Fähigkeit zur Generalisierung, Adaptionsfähigkeit und Reaktivität untersucht werden. Zusammengefasst werden folgende Versuche durchgeführt:

- Die Adaptionsfähigkeit wird geprüft, indem der Lernprozess im vorgegebenen Szenario durchgeführt wird, anschließend das Szenario verändert und wieder geprüft wird, wie lange der Agent für eine neue Lösung benötigt.
- Die Generalisierbarkeit wird geprüft, indem das in einem Szenario gewonnene Wissen in einem leicht veränderten Szenario angewendet wird.
- Die Reaktivität wird geprüft, indem nach einer Lernphase plötzlich neue Objekte / Hindernisse im Szenario auftauchen.

Die Versuche für die Analyse der Lernfähigkeit des DPA-RL-Agenten werden in der Realität durchgeführt und mithilfe von Videomaterial dokumentiert.

8.2.1 Adaptionsfähigkeit

Zur Bestätigung der Adaptionsfähigkeit wird ein einfaches Szenario 1 aufgebaut, siehe Abbildung 8.1. Der optimale Weg in Szenario 1 kann sowohl über den Durchgang 1 als auch über den Durchgang 2 erfolgen. Beide Durchgänge sind in der Abbildung gekennzeichnet. Der Agent muss nur wenige richtige Entscheidungen treffen. Anschließend wird das Szenario modifiziert, so dass der Agent umlernen muss um das Ziel zu erreichen. Die Anzahl der für das Umlernen benötigten Schritte, sowie das Verhalten des DPA-RL-Agenten wird untersucht.
In Abbildung 8.2 sind fünf Episoden dargestellt. Innerhalb der ersten drei Episoden hat der Agent die optimale Sequenz der Zustand-Aktions-Paare gelernt. Der Agent hat schnell gelernt den Durchgang 1 zu benutzen, um schnell zum Ziel zu gelangen. Die erste Episode hat 50 Sekunden gedauert, die 2. und 3. sind innerhalb der nächsten 2 Minuten erfolgt. Ab der 3. Episode verwendet der Agent das entscheidende Zwischenziel, das er nach der 2. Episode auf Durchgang 1 gesetzt hat, vergleiche die gekennzeichneten, gefahrenen Trajektorien in der Episode 2 und 3 in der Abbildung. Der Agent verfolgt schon in der 4. und 5. Episode den optimalen Weg. Die Episoden 4 bis 5 weisen fast den gleichen Ablauf auf.
Um die Adaptionsfähigkeit des Verfahrens in der Realität zu bestätigen, wird das Szenario 1 umgebaut. Der vom Roboter-Agenten bevorzugte Durchgang 1 wird zugestellt. Der Lernvorgang wird im erweiterten Szenario mit schon gewonnenem Wissen aus Szenario 1 fortgeführt. Der Roboter-Agent muss sein Wissen adaptieren, um das Ziel zu erreichen, siehe modifiziertes Szenario 1 in Abbildung 8.3.
Der Lernvorgang des Agenten nach der Modifizierung des Szenarios 1 ist für die ersten 5 Episoden in

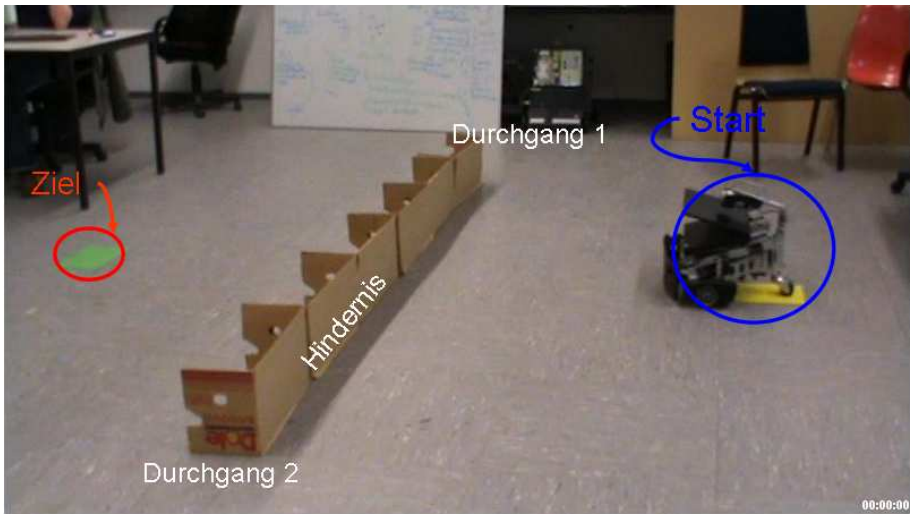


Abbildung 8.1: Szenario 1 mit zwei möglichen optimalen Wegen zum Ziel, entweder über Durchgang 1 oder 2.

Abbildung 8.4 und der weitere Verlauf der 6. bis 8. Episode in Abbildung 8.5 dargestellt. Insgesamt werden 7 Episoden zum Umlernen benötigt.
Zuerst wendet der DPA-RL-Agent das aus Szenario 1 gewonnene Wissen an, siehe Abbildung 8.4. Die erlernte Q-Funktion führt den Agenten zum in Durchgang 1 gesetzten Zwischenzielknoten. Der Agent versucht immer wieder dem Weg, der die größte Belohnung in Szenario 1 versprach, zu folgen. Die erste Episode dauert über drei Minuten, das Dreifache im Vergleich zur benötigten Zeit in der ersten Episode im nicht-modifizierten Szenario 1. Der Agent fährt zuerst zum gesetzten Zwischenziel. Dieses ist im modifizierten Szenario eine falsche Entscheidung. Der Agent versucht das Hindernis gegen den Uhrzeigersinn zu umfahren, diese Entscheidung hat den Agenten in vorherigen Episoden des Szenarios 1 zum Erfolg geführt. Im modifizierten Szenario führt die Entscheidung zum Misserfolg, so dass sich der Agent nach einiger Zeit entscheidet, wieder zum Ziel und nicht zum alten Zwischenziel zu fahren. An der Stelle, wo das Hindernis das Erreichen des Ziels stört, entscheidet sich der Agent wieder für die Aktion ”Hindernis links umfahren” (Hindernis im Uhrzeigersinn umfahren), die den Agenten wieder zur gleichen schon angefahrenen Station führt, so dass nach einiger Zeit die Aktion ”zum Ziel fahren” gewählt wird und sich die Situation wiederholt. Nachdem das Hindernis wieder im Weg steht, entscheidet sich der Agent zum dritten Mal für die Aktion ”Hindernis links umfahren”. Später entscheidet sich der Roboter an einer Stelle für die Aktion ”Hindernis rechts umfahren” (gegen den Uhrzeigersinn) statt der Aktion ”zum Ziel fahren”. Diese Aktion wird mehrere Male gewählt, so dass sich der Agent zum richtigen Durchgang 2 bewegt und aus der Falle herausfindet. In der 2. Episode wird das Zwischenziel, das im gesperrten Durchgang 1 liegt, wieder gewählt. An der Stelle, wo das Hindernis vor dem Agenten steht, wurde aber die richtige abstrakte Aktion ”Hindernis rechts umfahren” (Hindernis im Uhrzeigersinn umfahren) gewählt. Diese Aktion führt den Agenten zum richtigen Durchgang 2 und nach der 2. Episode wird das Ziel wieder erreicht. In der 3. Episode wird zuerst die Aktion ”zum Ziel fahren” gewählt, und an der Stelle, wo das Hindernis den Weg zum Ziel versperrt, wird die neu erlernte, richtige Entscheidung für die Aktion ”Hindernis rechts umfahren” getroffen. Die 3. Episode dauert 50 Sekunden. In der 4. und 5. Episode zeigt der Roboter-Agent, dass das neue in der 1. bis 3. Episode gewonnene Wissen noch nicht so stark ausgeprägt ist, wie das aus Szenario 1 vorhandene Wissen. Der Agent gerät in der 5. Episode wieder in einen Kreis, so dass die 5. Episode etwa 1 Minute und 50 Sekunden dauert.

Abbildung 8.2: Bildfolge für die ersten 5 Episoden in Szenario 1. Der Agent findet schon in der 1. Episode einen guten Weg zum Ziel. In der 2. Episode probiert der Agent einen alternativen Weg, kehrt aber zum bekannten Weg aus der 1. Episode um. Nach der 3. Episode ist das entscheidende Zwischenziel gesetzt. Die weiteren Episoden weisen nur kleinere Verbesserungen des gefundenen optimalen Weges auf.

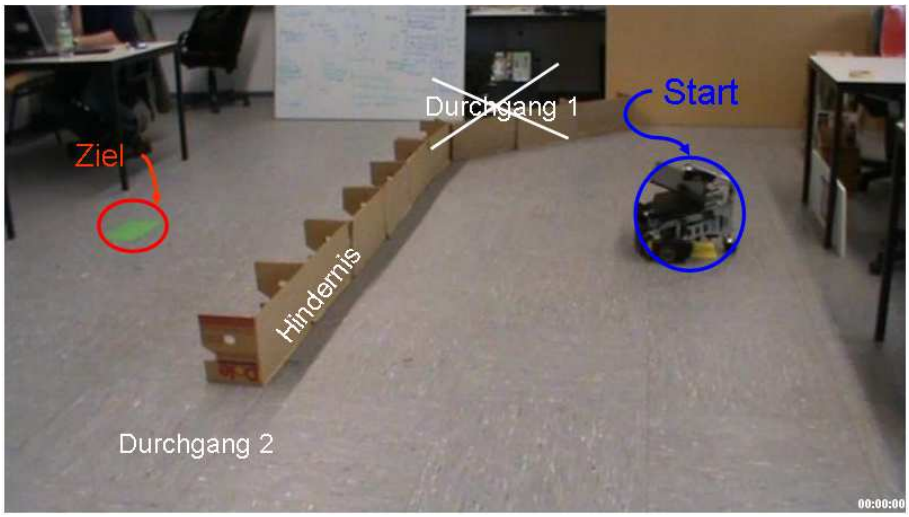


Abbildung 8.3: Modifiziertes Szenario 1: Der vom Agenten nach dem Lernprozess bevorzugte Durchgang 1 wurde zugestellt, der Agent soll sein Verhalten ändern, sonst gelangt er nicht zum vorgegebenen Ziel.

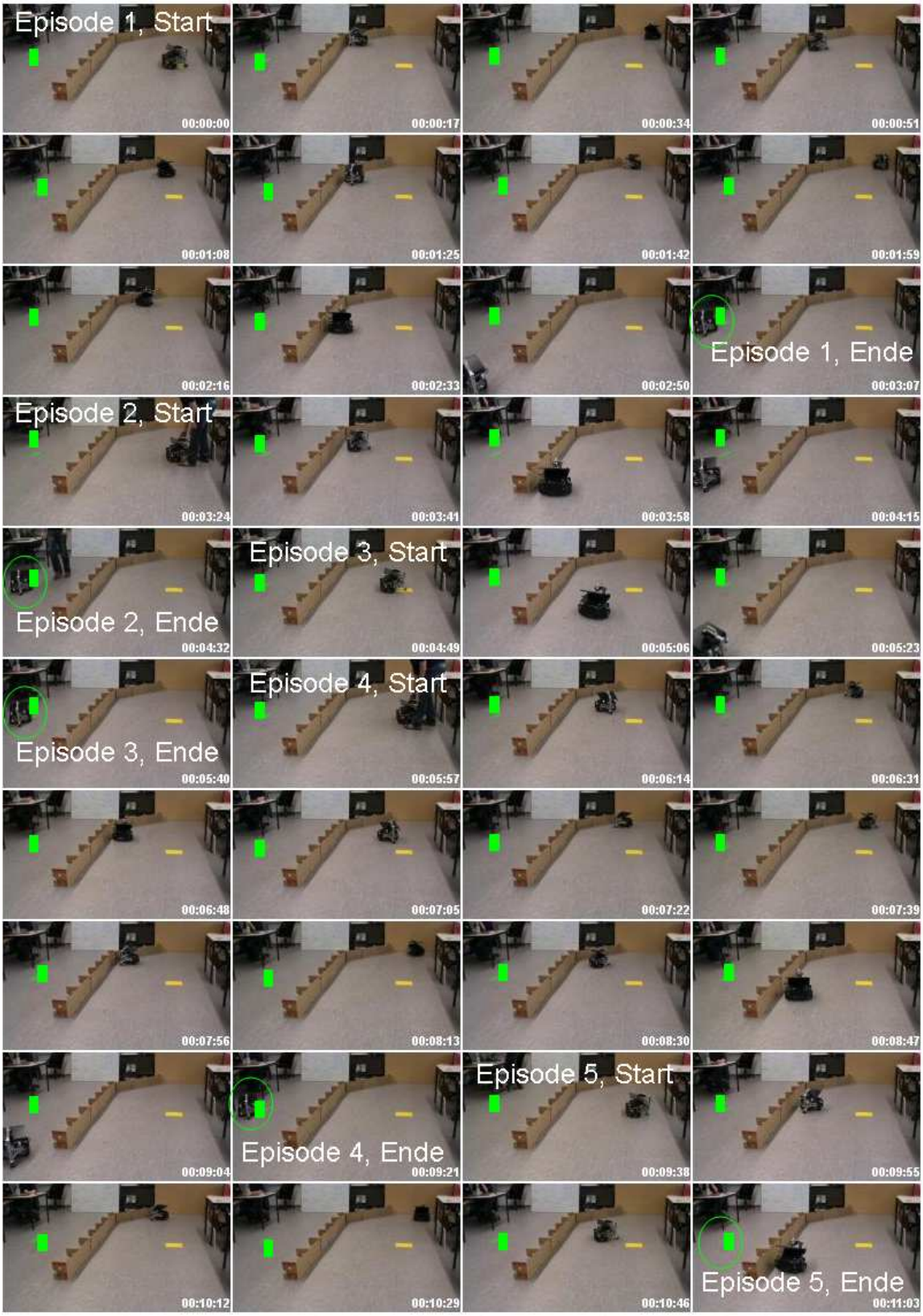


Abbildung 8.4: Bildfolge für die ersten 5 Episoden im modifizierten Szenario 1. Der Agent muss das erlernte Wissen berichtigen, und Durchgang 2 statt den bevorzugten Durchgang 1 zum Erreichen des Ziels wählen.

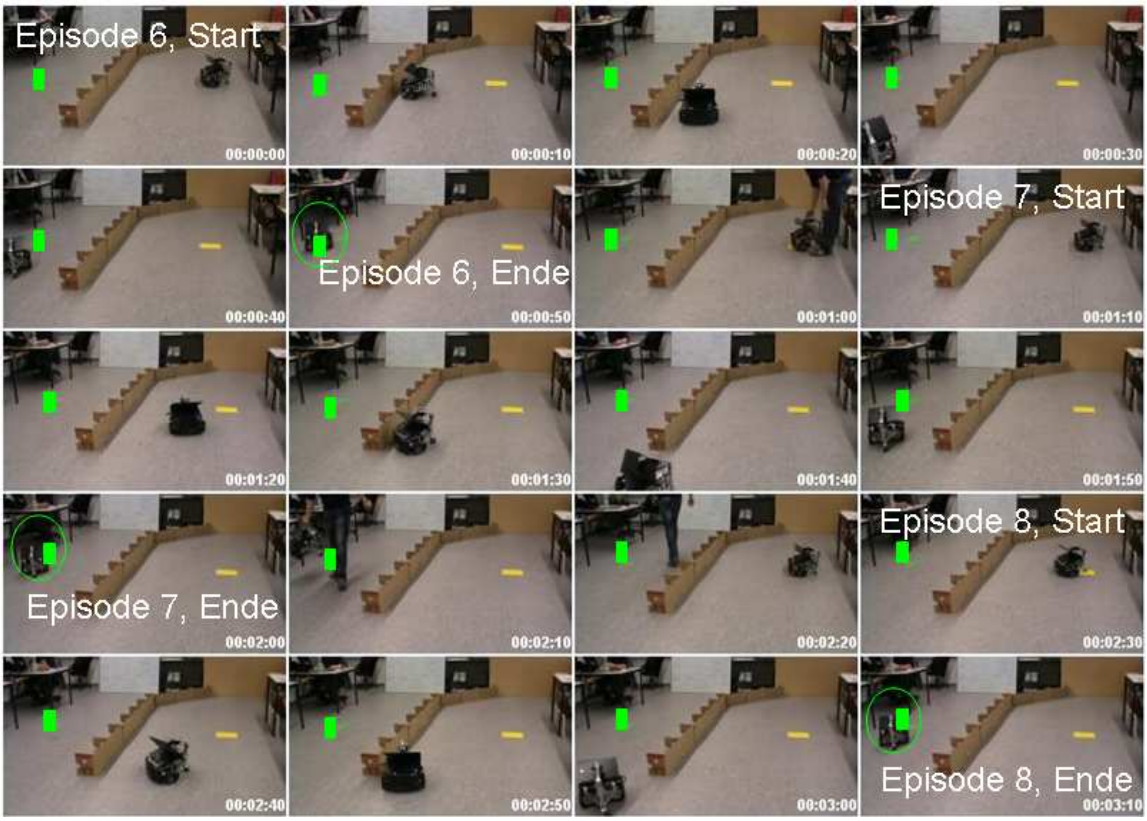


Abbildung 8.5: Bildfolge für die Episoden 6 bis 8 im modifizierten Szenario 1. Der Agent setzt sich ein neues Zwischenziel und versucht nicht mehr Durchgang 1 anzufahren.

Die Episoden 6 bis 8 weisen schon ein komplett umgelerntes Verhalten auf, siehe Abbildung 8.5. Erst ab der 6. Episode kommt das neue, richtig gesetzte Zwischenziel zur Geltung. Ab der 7. Episode entscheidet sich der Agent am Startpunkt zum neu bestimmten Zwischenziel zu fahren. Die Position des Zwischenziels ist aber nicht genau genug platziert, so dass der Agent ein zusätzliches Manöver benötigt, um seine weitere Fahrt fortzusetzen, dies erfolgt aber ohne weitere Schwierigkeiten. Der Agent versucht nicht mehr den optimalen Weg für das alte Szenario wieder auszuprobieren, sondern fährt zielstrebig zum Ziel in Richtung des neu gesetzten Zwischenziels in Durchgang 2.
Mit dem aufgestellten Versuchsaufbau und dem durchgeführten Lernvorgang wurde gezeigt, dass der Agent fähig ist das erlernte Wissen durch neues Wissen zu ersetzen. Der Adaptionsschritt ist aber mit Aufwand verbunden und kann sogar mehr Zeit in Anspruch nehmen als ein Lernvorgang ganz ohne Wissen am Anfang. Wenn der Agent an der aus der Vergangenheit bekannten Position angekommen ist, greift der Agent auf das veraltete Wissen zurück und wählt die falschen Aktionen aus, die zu langen Umwegen führen können.

8.2.2 Generalisierbarkeit

Die Generalisierbarkeit wird in einem komplexeren Szenario 3, siehe Abbildung 8.6, geprüft. Der aufgebaute Versuch besteht aus zwei Schritten. Beide Schritte werden in der Realität durchgeführt. Der erste Schritt besteht aus dem Lernen im komplexeren Szenario 3. Anschließend wird das Szenario 3 leicht verändert, und nur die grundlegende Hindernisform beibehalten, so dass der Agent im zweiten Schritt das gewonnene Wissen anwenden kann.
Das Szenario 3 ist ein komplexeres Szenario mit einer von der Startposition aus gesehen konkaven Hindernisform. Der Lernvorgang beinhaltet mehrere Entscheidungsknoten, was zu einem längeren Lernprozess als in Szenario 1 führt, vergleiche Bemerkung 8.1.1. Sechs Episoden werden zum Erlernen des optimalen Wegs benötigt. Das dargestellte Szenario 3 wird auch in der Untersuchung des Systems hinsichtlich Reaktivität, siehe Unterkapitel 8.2.3, angewendet, so dass der nach 6 Episoden erlernte Parametervektor für zwei weitere Versuche angewendet wird.
In den Abbildungen 8.7 und 8.8 ist der Lernprozess in Szenario 3 dargestellt. Am Anfang startet der

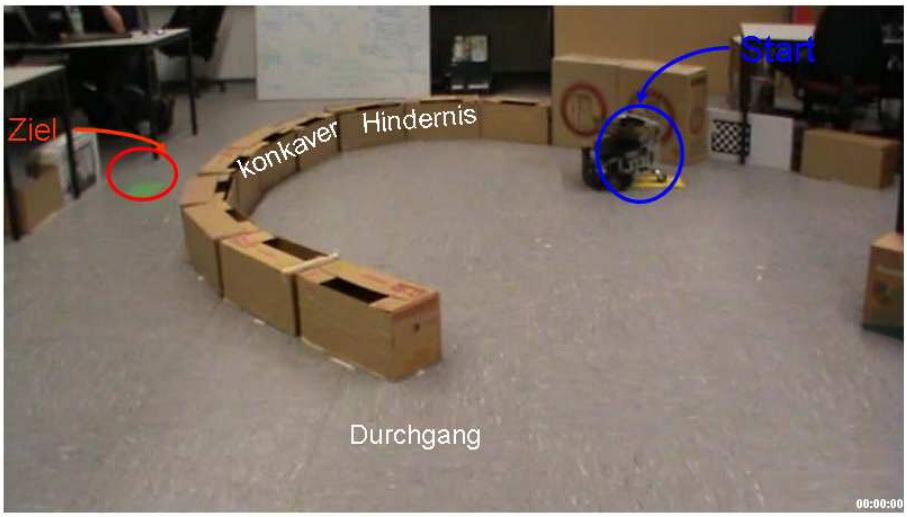


Abbildung 8.6: Szenario 3: Umgebung mit konkaver Hindernisform, konkav in Bezug auf den Startpunkt des Roboter-Agenten.

Agent ohne Wissen in die erste Episode. Der Agent wählt die Aktion ”zum Ziel fahren”, da nichts den

Weg zum Ziel versperrt. Der Agent wird solange durch die abstrakte Aktion ”zum Ziel fahren” zum Ziel navigiert, bis das Hindernis den direkten Weg zum Ziel versperrt. An der Stelle, wo die gewählte Aktion nicht mehr ausführbar ist, soll der Agent entscheiden, ob das Hindernis rechts oder links umfahren werden soll. Diese Stelle ist ein interner Zustand im Leistungselement und wird als ”wichtiger Entscheidungspunkt” bezeichnet. Die Entscheidung wird bei der ersten Episode zufällig getroffen, da alle Einträge in der Q-Funktion gleich Null sind. Hier entscheidet sich der Roboter-Agent für die Aktion ”Hindernis links umfahren”. Die getroffene Entscheidung führt zuerst zum Misserfolg. Der Agent dreht einen Kreis und gelangt wieder an den wichtigen Entscheidungspunkt. An der gleichen Stelle wird die neue Möglichkeit das Hindernis rechts zu umfahren ausprobiert. Dieser Versuch endet wieder im wichtigen Entscheidungspunkt. Der Agent macht immer größere Kreise, so dass nach einem drei Minuten langen Lernvorgang in der ersten Episode der Ausweg aus der konkaven Falle gefunden ist, siehe Abbildung 8.7. Die nächste Episode 2 dauert 1,6 Minuten. In dieser Episode wird die richtige Entscheidung für die Aktion ”Hindernis rechts umfahren” am Entscheidungsknoten getroffen, an der Stelle, wo die Aktion ”zum Ziel fahren” nicht mehr ausführbar ist.
In Abbildung 8.8 ist die 3. Episode dargestellt. Diese Episode dauert ungefähr vier Minuten. Die längere Erkundungszeit ist durch die falsche Entscheidung zu erklären, sich statt nach links nach rechts zu drehen, siehe roter Kreis, so dass der DPA-RL-Agent, der schon fast aus seiner Falle herausgefahren war, zurück in die konkave Falle umkehrt. Die Q-Werte werden dadurch versetzt, und neue Erkundungsschritte werden für das Umlernen der durch die falschen Entscheidungen entstandenen Fehler benötigt.

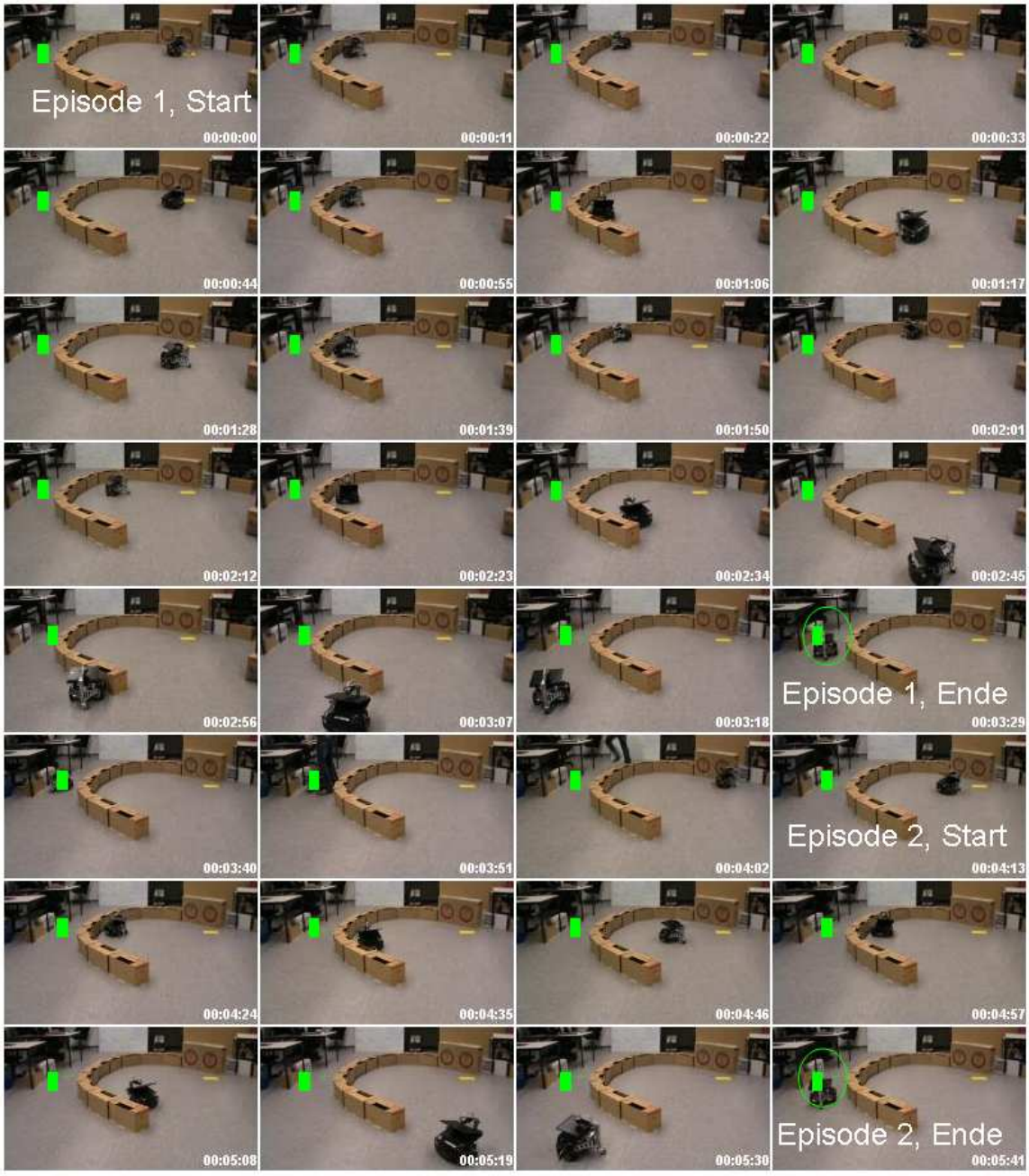


Abbildung 8.7: Bildfolge für das komplexere Szenario 3. Der Agent dreht in der 1. Episode mehrere Kreise von immer größerem Radius und findet schließlich den Ausgang aus der konkaven Falle. Die 2. Episode beinhaltet die vom Agenten gemachten Kreise, dauert aber halb so lange wie Episode 1.

Abbildung 8.8: Bildfolge der 3. Episode in Szenario 3, die länger dauert als die 1. Episode. Dieser lange Weg zum Ziel liegt an der falschen Entscheidung, die mit dem roten Kreis markiert ist.

Die 3. Episode weist aber einen entscheidenden Vorteil auf. Nach dieser Episode wird der wichtige Zwischenzielknoten eingesetzt, so dass der DPA-RL-Agent in den weiteren Episoden immer wieder den optimalen Weg mit der Verfolgung des eingesetzten Zwischenziels gefahren ist, vergleiche Abbildung 8.9. Hier lässt sich die bekannte Weisheit beobachten ”Aus Fehlern lernt man.”.
Um die angesprochene Generalisierbarkeit auf die Probe zu stellen, wird das Szenario 3 leicht verändert.

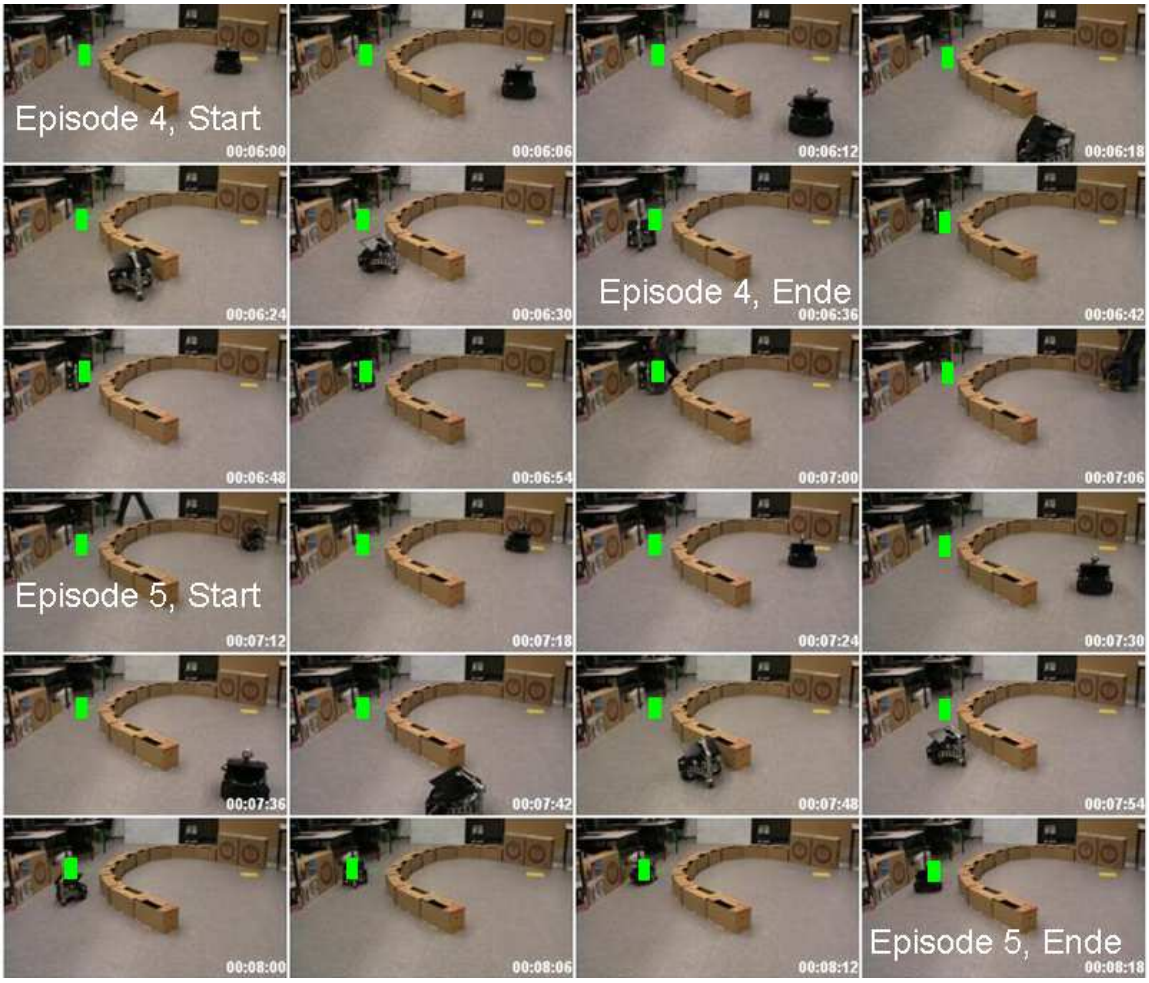


Abbildung 8.9: Bildfolge der 4. und 5. Episode in Szenario 3

Die Hindernisse werden leicht verschoben, so dass nur die konkave Hindernisform beibehalten wird, siehe Abbildung 8.10. Der Versuch wird in der veränderten Umgebung gestartet. Das in Szenario 3 nach fünf Episoden erworbene Wissen wird für diesen Versuch verwendet.
Der Agent hat die Veränderungen am Anfang gar nicht wahrgenommen, da ein Zwischenziel auf seinem optimalen Weg liegt, und der erlernte Weg zum Zwischenziel von den Änderungen im Szenario nicht betroffen ist, vergleiche die ersten zwei Zeilen in der Bildsequenz 8.11. Die Generalisierbarkeit dieses Verfahrens wird im weiteren Verlauf des Versuchs beobachtet. Die markanten Stellen sind durch rote Kreise gekennzeichnet, vergleiche Zeile vier und fünf. An diesen Stellen wird der Agent mit den verschobenen Hindernissen konfrontiert. Trotz der Verschiebungen der Hindernisse, also der Veränderung der lokalen Umgebung, werden vom Agenten die richtigen, in den vorherigen Episoden erlernten Entscheidungen getroffen. Der Agent bewegt sich zielstrebig zum vorgegebenen Ziel, siehe Abbildung 8.11. Ein ähnliches Szenario wird mit dem in ähnlichen Situationen erworbenen Wissen gelöst. Diese Tatsache lässt auf eine gute Generalisierbarkeit des Verfahrens schließen.

Abbildung 8.10: Szenario 3 mit verschobenen Hindernissen zur Bestätigung der Robustheit und Generalisierbarkeit des entwickelten Verfahrens

Der durchgeführte Versuch zeigt die Vorteile des entwickelten DPA-RL-Agenten, u.a. die Robustheit

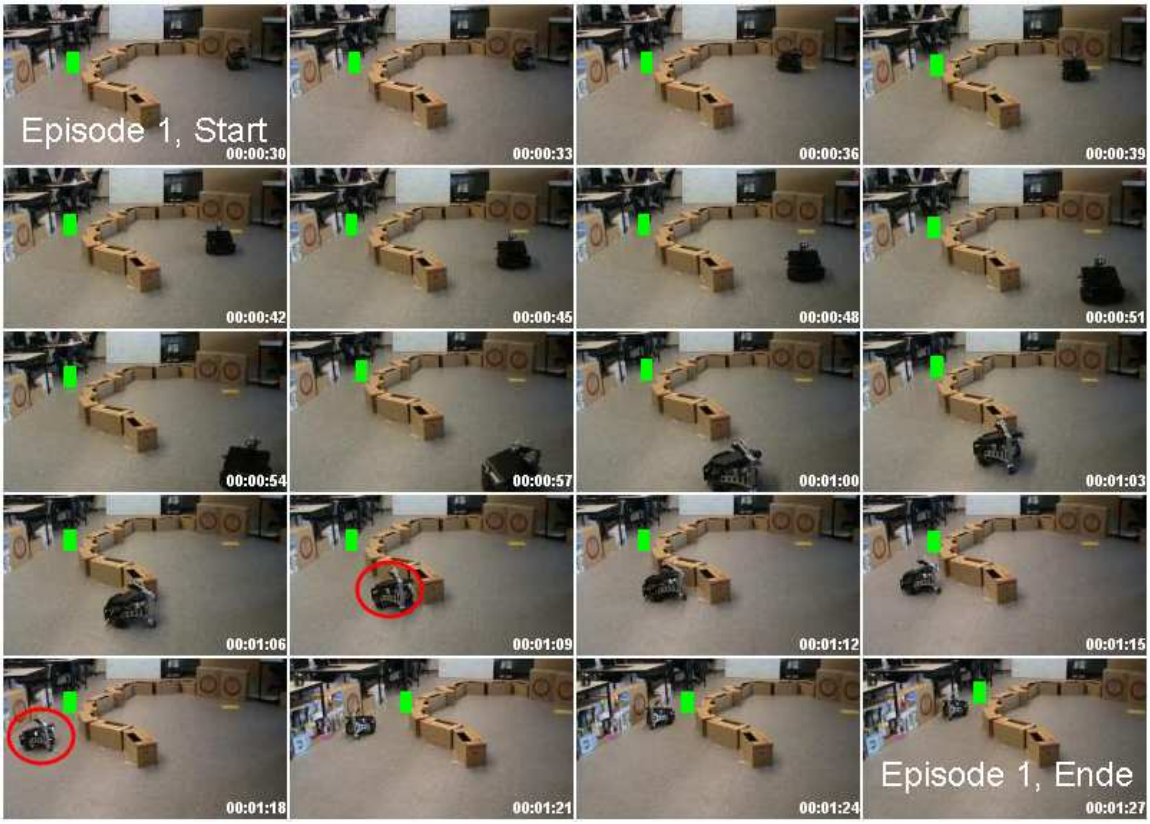


Abbildung 8.11: Episode 1 in Szenario 3 mit verschobenen Hindernissen. Der Agent bewegt sich am Anfang zielstrebig zum gesetzten Zwischenziel. Der weitere Weg des Agenten ist zwar von den Veränderungen in der Umgebung betroffen. Der Agent lässt sich aber auf seinem Weg zum Ziel kaum von den Veränderungen beeinflussen.

gegenüber leichten Veränderungen in der Umgebung. Die Stärke des entwickelten Konzepts der Zwi-

schenziele wird durch diesen Versuch bestätigt. Das gewünschte Verhalten wurde beobachtet. Leichte Veränderungen der Umgebung haben keine spürbare Auswirkung auf den Lernverlauf. Der Agent ist dazu fähig, die gewonnenen Erkenntnisse in ähnlichen Situationen anzuwenden.

8.2.3 Reaktivität

In diesem Abschnitt wird die Reaktivität des Systems geprüft. Sie ist durch ein Treffen von Entscheidungen in Echtzeit und eine situationsabhängige Änderung des vorliegenden Wissens gekennzeichnet.
Der Versuch wird zur Bestätigung der Reaktivität des entwickelten Systems durchgeführt. Das Szenario 3 aus Kapitel 8.2.2 und die Q-Funktion, die nach 5 Episoden aus dem durchgeführten Lernverlauf entstanden ist, werden dafür verwendet. Die Reaktivität des Systems wird durch eine kontrollierte Störung während des Episodenverlaufs geprüft. Diese Störung wird durch Eintreten einer Person vor den Agenten realisiert. Mit diesem Versuch wird neben der Reaktivität gleichzeitig auch die Adaptionsfähigkeit bzw. Generalisierbarkeit des Systems bestätigt.
In Abbildung 8.12 ist der Episodenverlauf dargestellt, in dem der Agent auf dynamische Hindernisse reagieren muss. Die entscheidenden Stellen in diesem Versuch sind die Stellen, an denen der Agent an der Fahrt zum gesetzten Zwischenziel gestört wird. Sie sind mit einem roten Kreis markiert. Der Weg zum Zwischenziel wird zwei Mal durch eine Person versperrt. Nach der ersten Störung, siehe 1. Bild in der 2. Zeile in Bildfolge 8.12, auf dem Weg zum Zwischenziel kehrt der Agent zurück in die konkave Falle. Dabei versucht der Agent die Person zu umfahren. Nachdem sie weg ist, versucht der Agent aus der Falle herauszufahren, indem das Zwischenziel wieder angefahren wird. Bei dem nächsten Versuch des Agenten das Zwischenziel anzufahren wird der Weg zum Zwischenziel wieder versperrt, vergleiche 2. Bild in der 4. Zeile. Die zweite Störung beim Erreichen des Zwischenziels hat eine größere Auswirkung auf den Lernprozess. Der Agent wählt, nachdem die Störung weg ist, nicht mehr die Aktion ”zum Zwischenziel fahren” aus, sondern zuerst die Aktion ”zum Ziel fahren” und anschließend die Aktion ”Hindernis links umfahren”. Der Agent hat diese letzte Aktion solange ausgeführt, bis die Ausfahrt aus der Falle, die durch eine konkave Hindernisform erzielt wird, erreicht ist, vergleiche Bildsequenz angefangen mit dem 4. Bild in der 5. Zeile bis zum 3. Bild in der 6. Zeile in Abbildung 8.12.
Trotz des Eingriffs an den entscheidenden Stellen ist der Agent den dynamischen Hindernissen erfolgreich ausgewichen. Wenn kein Hindernis mehr auf seinem Weg liegt, erreicht der Roboter-Agent sein Ziel schnell und zielstrebig.

8.3 Unterschiedliche Realisierungen abstrakter Aktionen

Die Besonderheit des DPA-RL-Agenten gegenüber dem RL-Agenten liegt in der Anwendung von Aktionen der höheren Abstraktionsebene. Die Möglichkeiten die abstrakten Aktionen unterschiedlich zu realisieren wurden bereits in Kapitel 6 beschrieben. In diesem Kapitel werden nun unterschiedliche Realisierungen der reaktiven Fähigkeiten, aus denen die abstrakten Aktionen zusammengestellt sind, miteinander verglichen. In diesem Abschnitt werden unterschiedliche Realisierungen der abstrakten Aktionen ”Hindernis rechts/links umfahren” und deren Einbindung in den DPA-RL-Agenten untersucht. Der Versuch wird in der Simulation für die Umgebung BC, siehe Abbildung 5.7, durchgeführt. Zum Vergleich beider Realisierungen werden die Werte für die intrinsische Energie der vom Roboter-Agenten gefahrenen Trajektorie angewendet. Diese Werte werden für die Trajektorien ermittelt, die nach 10 Episoden entstanden sind. Um die Abhängigkeit der Ergebnisse von der Trajektorienlänge zu beseitigen und eine Bewertung des reinen Trajektorienverlaufs zu ermöglichen, werden in jedem Versuch Trajektorien ähnlicher Länge ausgewählt.

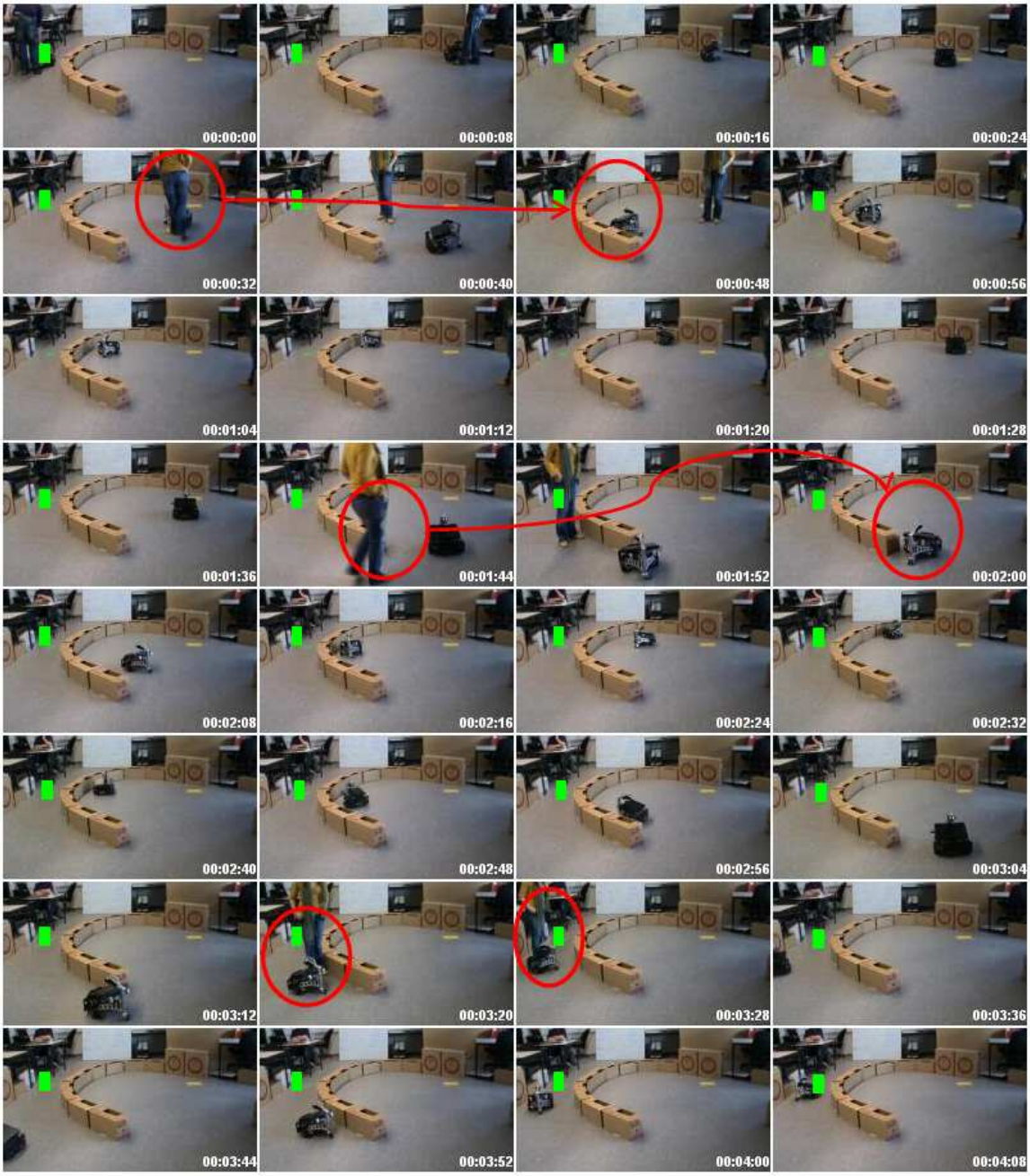


Abbildung 8.12: Während des Ausführens der Mission tritt eine Person an die entscheidenden Stellen vor den Roboter, so dass die erlernte optimale Politik nicht weiter durchgeführt werden kann. Der Agent reagiert adäquat auf die Störungen und dreht sich entsprechend um. Wenn der bekannte Weg frei wird, führt der Agent die erlernte Politik fort.

Die absoluten Werte der intrinsischen Energie der erzielten Trajektorie, die über 70 Versuche (also Trajektorien) bestimmt werden, sind in Tabelle 8.1 zusammengefasst.
Die intrinsische Energie der Trajektorien, die durch Anwendung der mit kNN realisierten, abstrakten Aktionen entstanden sind, ist um 11% besser als die intrinsische Energie der Trajektorien, die bei Anwendung der Realisierung mit Regelungstechnik entstanden sind, siehe Tabelle 8.2. Dieser Unterschied beider Trajektorien ist signifikant.

Tabelle 8.1: Auswirkung unterschiedlicher Realisierungen der abstrakten Aktionen des DPA-RL-Agenten in Umgebung BC, vergleiche Abbildung 5.7, absolute Werte, gewählte Trajektorien mit ähnlichen Energien

Implementierungsart	intrinsische Energie	Trajektorienlänge
PID-Regler	778.399	1200
kNN	692.222	1200

Zusammengefasst haben beide Realisierungen der abstrakten Aktionen ihre Vor- und Nachteile. Die An-

Tabelle 8.2: Auswirkung unterschiedlicher Realisierungen der abstrakten Aktionen des DPA-RL-Agenten in Umgebung BC, vergleiche Abbildung 5.7

Implementierungsart	Verbesserung der intrinsischen Energie	t-Test
kNN	11%	1

wendung der Regelungstechnik liefert stabil gute Ergebnisse und ist für die meisten Anwender eine vertraute Methode. Als Nachteil kann die Erfordernis eines Kalibrierschrittes angesprochen werden. Wenn sich die Konfiguration des Agenten ändert, wird eine Anpassung des erzielten PID-Reglers im Quellcode notwendig. Mit der NEAT-Methode erzielte kNN stellen eine weitere Möglichkeit der Repräsentation der gewünschten abstrakten Aktionen dar. Die Anwendung der kNN liefert zwar gute Ergebnisse, diese unterscheiden sich aber kaum von den mit einem PID-Regler erzielten Ergebnissen. Als Vorteile dieser Methode kann die intuitive Verständlichkeit der für das Training benötigten Szenarien und Fitnessfunktionen genannt werden. Als Nachteil kann die Notwendigkeit der Einbindung einer offenen Bibliothek und der Kenntnisse auf dem Gebiet der evolutionären Algorithmen genannt werden. Das Verfahren benötigt eine Einstellung mehrerer Parameter.
Beide Realisierungen der benötigten abstrakten Aktionen ”Hindernis rechts/links umfahren” werden im Offlinebetrieb erzielt. Es liegt keine Möglichkeit vor, die erstellten abstrakten Aktionen im Echtzeitbetrieb noch weiter zu verändern.
Die unterschiedlichen Realisierungen der abstrakten Aktionen haben wenig Auswirkung auf den gesamten Lernprozess. Die erzielten Ergebnisse für den Lernfortschritt (Lerngeschwindigkeit) sind fast gleich und hängen zum größten Teil von den getroffenen Entscheidungen des DPA-RL-Agenten ab.

8.4 Auswirkung der heuristischen Funktion im DPA-RL-Agenten

Allgemeingültige Möglichkeiten die Reinforcement-Learning-Methode zu beschleunigen werden in Kapitel 4 vorgestellt. Die Einbindung der vorgeschlagenen heuristischen Erweiterung wird für die DPA-RL-Architektur analysiert.
Der heuristische Einfluss hilft dann, wenn die Aussage der Heuristik mit dem optimalen Weg übereinstimmt. Er ist außerdem dann von Bedeutung, wenn mehrere Aktionen in einem Zustand zur Verfügung stehen. In der DPA-RL-Architektur werden die möglichen Aktionen durch den Agenten bestimmt und schon im Vorfeld aussortiert, so dass die Architektur selbst die Aufgabe der heuristischen Funktion übernimmt. Der Kompass-Sinn, vergleiche Abschnitt 2.5.2.3, der die Richtung zum Ziel angibt, übernimmt die Funktionalität der heuristischen Funktion, ergreift die Initiative, wenn es möglich ist, und führt den Agenten in Richtung des Ziels. Im zeitlichen Verlauf dieses Projektes hat die Untersuchung der heuristi-

schen Einflüsse und die Idee der Einbindung dieses Kontextwissens über die Zielposition entschieden zur Entwicklung der Konzepte des DPA-RL-Agenten beigetragen. Die wichtigste reaktive Fähigkeit ”zum Ziel fahren” stammt aus der Idee der Anwendung eines heuristischen Signals.
Die in diesem Kapitel vorgestellten Ergebnisse zeigen nur, dass eine weitere Verstärkung des heuristischen Signals in der Bildung der Strategie bzw. des Belohnungsmodells, keine spürbare Verbesserung der Lerngeschwindigkeit des Lernprozesses erzielt.
Die Umgebung BC wird an dieser Stelle gewählt, da diese Umgebung zum Vergleichen der Ergebnisse in den unterschiedlichen Kapiteln der vorliegenden Arbeit angewendet wird. Dieses Szenario ist groß und nicht einfach. Da der DPA-RL-Agent jetzt automatisch Kollisionen mit Gegenständen vermeidet, ist das Messkriterium Stoßzahl nicht mehr relevant.

8.4.1 ϵ^*-gierige Strategie

Die im DPA-RL-Agenten angewendeten abstrakten Aktionen können eine unterschiedliche Zeitdauer zur Ausführung benötigen. Der nächste interne Zustand s' des Agenten in einem Lernschritt, vergleiche Abschnitt 7.3.1.1, ist nicht vorhersehbar. Für die ϵ^*-gierige Strategie wird dieser Zustand aber benötigt, vergleiche Aktionswahl mit ϵ^*-gieriger Strategie aus Gleichung (4.1). Dieser Zustand s' kann durch eine fest vorgegebene, diskrete Aktion, die eine Berechnung einer Näherung des Zustands s' erlaubt, geschätzt werden. Es wird nur die Tendenz, ob eine Aktion durch die heuristische Funktion besser als die andere ist, gemessen, so dass die Näherung ausreichend ist. An der Stelle der abstrakten Aktion ”Hindernis rechts umfahren” wird die Aktion (rechts $+90°$, eine Einheit im Orientierungsvektor) für die Schätzung des Zustands s' verwendet. An der Stelle der abstrakten Aktion ”Hindernis links umfahren” wird die Aktion ($-90°$, eine Einheit in Richtung des neu bestimmten Orientierungsvektors) für die Schätzung des Zustands s' angewendet. An der Stelle der abstrakten Aktion ”zum Ziel fahren” wird die Aktion ($0°$, eine Einheit in Richtung des neu bestimmten Orientierungsvektors) für die Schätzung des Zustands s' verwendet. Die Einheit wird durch eine Konstante ($10\ cm$) vorgegeben. Die Einbindung der ϵ^*-gierigen

Tabelle 8.3: ϵ^*-gierige Strategie für Umgebung BC (Abbildung 5.7). Verwendeter Parametersatz: $\alpha = 0.2, \epsilon = 0.05, \gamma = 0.95, \lambda = 0.5, d = 50, Radius = 50, Winkel = 45°$

ϵ^*-gierige Strategie	LP	t-Test LP
ja	2.58%	0

Strategie hat keinen aussagekräftigen Einfluss auf den Lernfortschritt (Lerngeschwindigkeit), vergleiche Tabelle 8.3.

8.4.2 SARSA*-Algorithmus

Die Einbindung der heuristischen Funktion im Belohnungsmodell und somit der direkte Einfluss der heuristischen Funktion auf den Lernprozess ist in seiner Auswirkung komplexer und besitzt wichtige Parameter zur Einstellung der Stärke des heuristischen Anteils des Belohnungsmodells. Die optimale Parametrisierung des heuristischen Einflusses im SARSA*-Algorithmus, siehe Kapitel 4.4, wird in Umgebung BC untersucht.
In diesem Kapitel wird die Hypothese über einen positiven Einfluss der heuristischen Funktion im Belohnungsmodell auf den DPA-RL-Agenten geprüft. Die Versuchsreihen werden in der Simulation mit unterschiedlichen Parametern durchgeführt, um die statistische Aussagekraft der Ergebnisse gewährleisten zu können. In Tabelle 8.4 ist der gesamte Überblick über die Ergebnisse dargestellt. Dabei gibt die

erste Spalte den verwendeten Wert für den Einflussparameter k an, siehe Definition der heuristischen Funktion in Gleichung 4.4. Dieser Einfluss wird von $k = -0.1$ mit einer kaum spürbaren Ausprägung bis $k = -2.0$ mit einer starken Ausprägung definiert. Die in der Tabelle dargestellten Zahlen geben die prozentuale Differenz zum reinen Verfahren ohne Einfluss der heuristischen Funktion im Belohnungsmodell, also $k = 0$, an. Positive Werte zeigen eine Verbesserung, negative Werte eine Verlangsamung des Lernprozesses an. Der für den Versuch genutzte Parametersatz ist in der Tabellenüberschrift angegeben. Aus der Tabelle wird ersichtlich, dass die heuristische Funktion im Lernprozess keinen positiven Ein-

Tabelle 8.4: SARSA*-Algorithmus, angewendet in Umgebung BC, siehe Abbildung 5.7. Verwendeter Parametersatz: $\alpha = 0.2, \epsilon = 0.05, \gamma = 0.95, \lambda = 0.5, d = 50, Radius = 50, Winkel = 45$

k	LP	t-Test LP
-0.1	0.03%	0
-0.2	-3.83%	0
-0.5	-3.84%	0
-0.7	-9.4%	1
-1.0	-9.43%	1
-2.0	-16.94%	1

fluss auf den Lernfortschritt (Lerngeschwindigkeit) hat. Wenn der Einfluss des heuristischen Signals zu stark gewählt wird, kann es sogar zu Störungen kommen, siehe die signifikante Verschlechterung in den Zeilen 4, 5 und 6 im Vergleich zum Ergebnis ohne jeglichen Einfluss der heuristischen Funktion im Belohnungsmodell. Die heuristische Funktion ist schon indirekt in Form des Kompass-Sinns in der DPA-RL-Architektur vorhanden. Die Anzahl der Zustände im Zustandsraum ist sehr gering, die erlernten Ketten aus Entscheidungen sind kurz. Es lässt sich vermuten, dass in Umgebung BC keine Verstärkung des heuristisches Wissens über den Lernprozess benötigt wird.

8.5 Analyse und Ausblick

Die Vorteile des entwickelten Systems sind die nachgewiesene Adaptionsfähigkeit, die Fähigkeit zur Generalisierung und die Reaktivität. Das System ist modular aufgebaut, d.h. die einzelnen Bestandteile des Systems sind austauschbar. Diese Möglichkeiten werden auch in den Untersuchungen ausgeschöpft. Unterschiedliche Realisierungen der abstrakten Aktionen werden untersucht, ohne dass gravierende Unterschiede festgestellt werden, so dass je nach Verfügbarkeit der Realisierungen der reaktiven Fähigkeiten jede Variante angewendet werden kann. Der PID-Regler ist relativ gut. Wenn ein Spezialist verfügbar ist, der die benötigte Kalibrierung durchführen kann, ist diese Realisierung der abstrakten Aktion zu bevorzugen. Wenn die NEAT-Methode zur Verfügung steht, und ein Spezialist die richtige Fitnessfunktion erstellt, können auf diese Weise festgelegte, künstliche Neuronale Netze auch angewendet werden.
Der vorgeschlagene Ansatz ist sehr flexibel. Er kann in den unterschiedlichsten Szenarien, z. B. in einer Wohnung oder einer Werkstatt angewendet werden. Der Ansatz benötigt am Anfang kein Wissen und keine Karte von der Umgebung. Innerhalb weniger Episoden weist der Agent schon ein sehr gutes Verhalten auf und erreicht das vorgegebene Ziel mit einer sehr guten Trajektorie. Eine der wenigen Voraussetzungen an die Umgebung, in der die Navigation durchgeführt werden soll, ist, dass das Bewegungsareal des Agenten auf einer Ebene liegt. Der Agent kann z. B. noch nicht Treppen steigen oder einen Fahrstuhl benutzen. Wenn diese Fähigkeiten aber vorhanden wären, könnten sie in die flexible Struktur ohne großen Aufwand eingebaut werden. Die Konfiguration (Hardware) des Roboter-Agenten

selbst kann ohne Schwierigkeiten ausgetauscht werden. In diesem Fall müssen nur die entsprechenden reaktiven Fähigkeiten an die vorgegebene Konfiguration aus Sensoren und Aktuatoren angepasst werden. Der evolutionäre Ansatz bietet die Möglichkeit diesen Schritt durch eine Trainingsphase im Offlinebetrieb durchzuführen, so dass keine aufwendige Kalibrierphase durchgeführt werden muss.
Der Nachteil des Systems liegt in den verwendeten Sensoren. Sie senden und empfangen Strahlen, die nur punktuelle Messungen in der Umgebung vornehmen, diese Tatsache kann zu Problemen führen[47]. Zum Beispiel bereiten Tischbeine in der Laborumgebung dem DPA-RL-Agenten größere Probleme. Da die Strahlen der verwendeten IR-Sensoren nicht den kompletten Raum um den Agenten herum abtasten, sondern nur sieben diskrete Stellen als Input verwenden, sieht der entwickelte Agent Tischbeine nur in seltenen Fällen, eher zufällig. Daher sollten dünne Hindernisse mit kleiner Fläche in den Szenarien im Vorfeld abgedeckt werden, damit der Agent die Hindernisse mit seiner diskreten Sensorabtastung sieht.
Die Anzahl der verwendeten abstrakten Aktionen im DPA-RL-Agenten ist nicht so groß, dies führt zu einer eingeschränkten Anwendbarkeit. Aber dadurch erfolgt der Lernprozess in Echtzeit. Das Gleichgewicht zwischen der Anzahl der möglichen abstrakten Aktionen, der Komplexität des verwendeten Optionenbaums und dem Lernfortschritt (Lerngeschwindigkeit), um das System in Echtzeit betreiben zu können, soll beibehalten werden. Dies ist ein typisches Bias-Varianz-Dilemma in der künstlichen Intelligenz. Auf der einen Seite wird ein nicht zu starres Modell gebraucht, was genügend Möglichkeiten bietet, um das gewünschte Verhalten auszubilden. Auf der anderen Seite ist ein zu flexibles Gebilde aus Zeitgründen und wegen Überanpassungsproblemen unerwünscht.
Die dynamische Erweiterung des Aktionsraumes kann auch anders realisiert werden. Die Bestimmung von Zwischenzielen kann unter Anwendung des graphentheoretischen Ansatzes zur Suche nach zusammenhängenden Komponenten realisiert werden. Die sogenannte Skelettierung des Graphen [Efimenko, 2011] kann auch zur Gewinnung von Zwischenzielen verwendet werden.

[47] Die Lösung für dieses Problem wird bereits vorbereitet. Am Lehrstuhl Intelligente Systeme wird ein Radarsystem mit einem in der Ebene beweglichen IR-Sensor entwickelt. Dieses System kann an die Steuerungskomponente angebunden werden, so dass das geschilderte Problem vermieden werden kann, vergleiche [Zhigun, 2011].

Kapitel 9

Fazit und Ausblick

Die vorliegende Arbeit beinhaltet neue Ideen für die Erweiterung des verbreiteten SARSA-Algorithmus, einen theoretisch fundierten Hintergrund, unter welchen Rahmenbedingungen die Erweiterung positiv oder zumindest nicht negativ ist, und die Anwendung der Lernmethode für echte Probleme auf echter Hardware in realen Umgebungen. Die vorgeschlagene Erweiterung des SARSA-Algorithmus kann ohne großen Aufwand für jede Art des Problems angewendet werden. Mithilfe der für die Erweiterung aufgestellten Bedingungen können die Abschätzungen schon im Vorfeld durchgeführt werden.
Das Navigationsproblem ist für die autonome Robotik von großer Bedeutung. Für die Lösung des Navigationsproblems werden viele Probleme behandelt, die auch von großer Wichtigkeit für das ganze Gebiet der künstlichen Intelligenz (KI) sind. Aus diesem Grund wird diese Arbeit als ein Beitrag zur modernen KI angesehen.

9.1 Erweiterung des Lernalgorithmus

In der vorliegenden Arbeit wird die Erweiterung des SARSA-Algorithmus um heuristisches Wissen vorgeschlagen. Zwei Möglichkeiten, wie das heuristische Wissen eingebunden werden kann, sind zur Geltung gekommen: das heuristische Wissen im Lernprozess (SARSA*-Lernalgorithmus) und heuristisches Wissen bei der Aktionswahl (ϵ^*-gierige Strategie). Das heuristische Wissen ist als normierte, parametrisierte, heuristische Funktion formuliert. Die Parametrisierung wird in Abhängigkeit vom angewendeten Belohnungsmodell und den im Lernprozess verwendeten Parametern (γ, λ) definiert. Die theoretische Herleitung der richtigen Parametrisierung der heuristischen Funktion wird durchgeführt. Die erzielten theoretischen Grenzwerte des zulässigen Intervalls des heuristischen Einflussparameters werden mithilfe von Grenzwerten, die mit einer vollständigen Suche erzielt werden, für beide Fälle geprüft. Die Auswirkung der vorgeschlagenen Erweiterung des SARSA-Algorithmus wird für drei unterschiedliche Aufgaben gemessen: das Gitterweltproblem, das Mountain-Car-Problem und das Navigationsproblem.
Die Konstruktion der heuristischen Funktion ist für den SARSA*-Algorithmus überall gleich und wird als Abstandsfunktion bis zum Zielzustand definiert.
Das Gitterweltproblem ist übersichtlich, beinhaltet einen diskreten Zustandsraum mit wenigen Zuständen und einen Aktionsraum mit vier möglichen Aktionen. Für dieses Problem werden zwei Szenarien konstruiert. Bei der Konstruktion dieser Szenarien wird auf die Entstehung von Widersprüchen zwischen dem optimalen Weg und dem durch die heuristische Funktion vorgeschlagenen Weg geachtet. In diesen Szenarien wird eine Verbesserung der Lerngeschwindigkeit für den SARSA*-Algorithmus um 40% bzw. 50% im Vergleich zum gewöhnlichen SARSA-Algorithmus erzielt. Diese Ergebnisse sind mit einer optimalen Parametrisierung der heuristischen Funktion im SARSA*-Algorithmus entstanden. Die optimalen Parameter der heuristischen Funktion unterscheiden sich für beide Szenarien, was durch die Besonderheiten im Aufbau der Szenarien zu erklären ist.
Das Mountain-Car-Problem ist ein Benchmark-Problem für Reinforcement-Learning-Methoden. Die-

ses Problem beinhaltet einen stetigen Zustandsraum, der aus zwei Teilen besteht, dem Weg und der Geschwindigkeit. Für die heuristische Funktion im SARSA*-Algorithmus wird nur der Weg-Anteil berücksichtigt. Je nach Parametrisierung des Lernprozesses wird für den SARSA*-Algorithmus eine Verbesserung um 8% ($\lambda = 0$), $4 - 6\%$ ($\lambda = 0.5$) und $6 - 10\%$ ($\lambda = 0.95$) erzielt. Dabei ist nur eine Summenoperation als zusätzlicher Aufwand notwendig. Die Ergebnisse sind gegenüber kleinen Änderungen des heuristischen Einflussparameters stabil.
Im Navigationsproblem ist der Zustandsraum stetig und besteht aus drei Teilen, der zweiteiligen Position des Agenten im Raum und seiner Orientierung. Die heuristische Funktion für den SARSA*-Algorithmus hängt auch in diesem Fall nur von der Position und nicht von der Orientierung des Agenten ab. Eine Verbesserung der Lerngeschwindigkeit wird für drei konstruierte Szenarien gemessen. Die optimale Parametrisierung der heuristischen Funktion hängt vom Aufbau der Umgebung ab. Eine Verbesserung um 10% für Umgebung 1, 15% für Umgebung 2 und sogar fast 25% für Umgebung 3 wird bei der Anwendung des SARSA*-Algorithmus gemessen.
Die ϵ^*-gierige Strategie reagiert empfindlicher auf die Definition der heuristischen Funktion. Für die Bildung der ϵ^*-gierigen Strategie hat die Definition der auf der Position des Agenten basierenden heuristischen Funktion für das Erzielen aussagekräftiger Ergebnisse im Mountain-Car-Problem und Navigationsproblem nicht ausgereicht. Für die ϵ^*-gierige Strategie im Navigationsproblem musste die heuristische Funktion in Abhängigkeit von der Orientierung definiert werden. Für das Mountain-Car-Problem wird die heuristische Funktion in Abhängigkeit von der Startposition anstatt der Zielposition des Agenten definiert.
Die Anwendung der ϵ^*-gierigen Strategie im Gitterweltproblem hat eine Verbesserung von bis zu 12% erreicht.
Für die Bestimmung der optimalen Strategie im Mountain-Car-Problem wird der SARSA-Algorithmus beim Lernen und die ϵ^*-gierige Strategie bei der Bestimmung der nächsten Aktion angewendet. Dabei wird eine Verbesserung von $4 - 5\%$ ($\lambda = 0.95$) erzielt. Die Ergebnisse sind gegenüber kleinen Veränderungen der heuristischen Einflussparameter stabil.
Im Navigationsproblem wird eine Verbesserung von bis zu 23% für Umgebung 2 und bis zu 18% für Umgebung 3 erzielt. Die Ergebnisse sind gegenüber kleinen Veränderungen der heuristischen Einflussparameter stabil. Die Existenzberechtigung für die Idee der Einbindung der heuristischen Funktion im Lernprozess (SARSA*-Algorithmus) und in der Bildung der Strategie (ϵ^*-Strategie) wird durch die erzielten Ergebnisse nachgewiesen. Die Justierung des heuristischen Einflussparameters soll nicht zu fein sein, es reicht nur eine ungefähre Abschätzung, um schon gute Ergebnisse zu erzielen. Die Verbesserung von bis zu 50% im Gitterweltproblem weist auf ein großes Potenzial der vorgeschlagenen Idee für den SARSA*-Algorithmus hin. Die Verbesserung von bis zu 23% im Navigationsproblem weist auch auf ein großes Potenzial der vorgeschlagenen Erweiterung mit einer ϵ^*-gierigen Strategie hin.

9.2 Bildung der abstrakten Aktionen

Zur Gewinnung der reaktiven Fähigkeiten und daraus aufgebauten abstrakten Aktionen werden in der vorliegenden Arbeit unterschiedliche Methoden angewendet und miteinander verglichen. Die erzielten reaktiven Fähigkeiten sind autonom, so dass der Agent und seine Umgebung, insbesondere Menschen in seiner Nähe, geschützt werden. Die benötigten reaktiven Fähigkeiten werden in einer realitätsnahen Simulation erzeugt, so dass kein weiterer Anpassungsschritt für die erzielte reaktive Fähigkeit in der Realität durchgeführt werden muss.

Für den Aufbau der benötigten reaktiven Fähigkeit ”Hindernis umfahren” werden zwei Möglichkeiten eingeführt. Je nach Randbedingungen und Verfügbarkeit der Instrumente und Spezialisten können die Kontrollfunktionen, aus denen die gewünschte reaktive Fähigkeit erzeugt wird, sowohl durch künstliche neuronale Netze (kNN), die mit dem evolutionären Ansatz erzeugt werden, als auch durch konventionelle Regelkreise repräsentiert werden.
Die Bestimmung der Kontrollfunktion mithilfe eines evolutionären Ansatzes ist für den Anwender viel intuitiver, erfordert keine Programmierkenntnisse und benötigt keine Aufbereitung der Sensorwerte. Es genügt die Fitnessfunktion aufzustellen und die passenden Szenarien für die Simulation zu konstruieren, um die gewünschten Fähigkeiten zu erzeugen. Eine Veränderung der Roboterkonfiguration im Fall der Anwendung der evolutionären Algorithmen (EA) ist weniger schmerzhaft als bei der Anwendung der Regelkreise. Die Konfiguration des Roboters soll zwar für beide Fälle verändert werden, allerdings bleiben die Anwendung der EA zur Bestimmung der kNN, die Szenarien und die Fitnessfunktion gleich.
Die Einbindung der mit einem PID-Regler entwickelten Kontrollfunktion benötigt die Aufbereitung der Sensorwerte und einen Kalibrierschritt. Um die gewünschte reaktive Fähigkeit zu erstellen, sollen die durch PID-Regelkreise repräsentierten Kontrollfunktionen miteinander verknüpft werden. Für den Aufbau der Kontrollfunktion mit Regelkreisen wird keine spezielle Software benötigt, wie es bei Anwendung der künstlichen neuronalen Netze der Fall ist. Die Regelkreise sind eine sehr verbreitete und für die meisten Nutzer von Robotern vertraute Technik.
Beide Methoden können für die Bestimmung der benötigten reaktiven Fähigkeiten angewendet werden, beide Realisierungen erzielen ähnlich gute Ergebnisse.

9.3 Anwendung der Lernalgorithmen im Navigationsproblem

9.3.1 RL-Agent

Der Reinforcement-Learning-Agent (RL-Agent) findet die Lösung für das Navigationsproblem im Onlinebetrieb mithilfe des SARSA-Algorithmus. Diese entwickelte Agentenart dient zu Analysezwecken und als Prototyp des Doppelpass-Architektur-RL-Agenten (DPA-RL-Agenten). Der RL-Agent löst das gestellte Navigationsproblem und kann dynamische Hindernisse umfahren. Er ist ungefährlich für die sich in den Räumen aufhaltenden Menschen, da ein Schutzmechanismus in Gefahrensituationen zur Geltung kommt. Der Schutzmechanismus wird durch reaktive Fähigkeiten repräsentiert. Der RL-Agent hat drei Aktionen in seinem Aktionsraum, und der Zustandsraum besteht aus der Position und Orientierung des Agenten. Die große Anzahl der vom RL-Agenten für das Lernen benötigten Episoden ist für die Anwendung der Szenarien mit echten Räumlichkeiten nicht akzeptabel.
Der Schutzmechanismus erfüllt seinen Zweck, er beschützt den Agenten vor Kollisionen mit einer Wand. Dabei kann dieser Mechanismus auch bedeutend zur Beschleunigung des Lernprozesses beitragen.
Die Einführung des Schutzmechanismus liefert eine Verbesserung der Stoßzahl um etwa 60% in vier konstruierten Szenarien. Die Verbesserung des Lernfortschritts ist nicht so stabil und hängt von der Beschaffenheit der Umgebung ab. Je größer und komplexer die Umgebung ist, desto kleiner ist der Einfluss des Schutzmechanismus auf die Lerngeschwindigkeit. Zum Beispiel wird mit einer optimalen Parametrisierung in Umgebung 1 eine Verbesserung der Lerngeschwindigkeit um 31.98% gemessen. Dagegen wird im realitätsnahen Szenario Umgebung BC eine Verbesserung um 5.73% gemessen.
Es konnte gezeigt werden, dass die Einbindung der zusätzlichen Informationen über den Aufbau des Szenarios den Lernprozess beschleunigen und die Anzahl der Kollisionen des Agenten verringern kann. Dieser Schritt wird durch die Bildung eines Kraftfelds auf Grundlage der vorhandenen Informationen über die Beschaffenheit der Umgebung realisiert. Das erzielte Kraftfeld wird durch den Projektions-

schritt in eine Vorinitialisierung der Bewertungsfunktion umgewandelt. Der entwickelte Vorschlag für die Erweiterung des SARSA- Algorithmus kann auch für eine weitere Adaption der erzielten Kraftfelder angewendet werden. Die richtige Relation zwischen dem Belohnungsmodell und der Parametrisierung dieses Schrittes ist von großer Bedeutung. Bei den meisten Versuchen wird eine Korrelation zwischen der Verbesserung beider Messkriterien beobachtet.
Die Verbesserung des Lernfortschritts bei der Einbindung des Vorinitialisierungsschrittes in den SARSA-Algorithmus liegt zwischen knapp 10% für Umgebung 2 und fast 30% in Umgebung 3 für den besten Parametersatz und hängt von der Beschaffenheit der Umgebung ab.
Der Vorteil des vorgeschlagenen Vorinitialisierungsschrittes liegt in der Fähigkeit des angewendeten SARSA-Algorithmus das ungenaue Wissen in Echtzeit zu adaptieren, was die gewöhnlichen Planungsalgorithmen ohne Erweiterungsschritt nicht erlauben. Die Qualität der angewendeten Informationen steht in einer schlechten Relation zur erzielten Verbesserung der Lerngeschwindigkeit.

9.3.2 DPA-RL-Agent

Der entwickelte DPA-RL-Agent stellt die endgültige Lösung für das gestellte Navigationsproblem dar. Diese Art des Agenten erlernt die optimale Kette der abstrakten Aktionen, die den Agenten zum Ziel führt, baut sein Umgebungsmodell auf, erweitert dynamisch seine Aktionsmenge und kann ohne den Simulationsschritt in den realen Szenarien angewendet werden. Die Besonderheit dieser Agentenart liegt im Zusammenspiel der Doppelpass-Architektur und des SARSA-Algorithmus, der mit einer Palette von abstrakten Aktionen arbeitet. Der entwickelte Agent ist in unterschiedlichsten Szenarien ohne Vorbereitungsschritte anwendbar.
Der DPA-RL-Agent beinhaltet als Basis die Doppelpass-Architektur, die ein paar Besonderheiten aufweist. Als zentrale Datenstruktur dient ein hierarchisch aufgebauter Baum. Dabei sind zwei Kontrollprozesse für die Ausführung der Mission verantwortlich. Die beiden Prozesse sind zwei unabhängige Top-Down-Prozesse, die den gesamten Optionenbaum von der Wurzel bis zu den Blättern des Baumes traversieren. Der Planungsprozess, der einen Plan für die Ausführung erstellt, und der Ausführungsprozess, der den aufgestellten Plan ausführt, schalten einander ein. Der aufgebaute Optionenbaum beinhaltet als Wurzelknoten den eingeführten RL-Choice-Knoten, der die möglichen abstrakten Aktionen als Kindknoten beinhaltet. Die Aufgabe solcher RL-Choice-Knoten liegt im Erlernen der optimalen Strategie. Die optimale Strategie schaltet die verfügbaren abstrakten Aktionen in Abhängigkeit von der aktuellen Situation ein, um den Agenten so schnell wie möglich zum Ziel zu navigieren. Die abstrakten Aktionen werden aus den vorhandenen reaktiven Fähigkeiten aufgebaut. Am Anfang sind nur drei mögliche Aktionen vorhanden, ”zum Ziel fahren”, ”Hindernis rechts umfahren” und ”Hindernis links umfahren”. Während der Ausführung der Mission kann der Aktionsraum erweitert werden, so dass dem RL-Choice-Knoten zusätzliche Kindknoten hinzugefügt werden.
Die Vorteile des entwickelten DPA-RL-Agenten sind die nachgewiesene Adaptionsfähigkeit, seine Fähigkeit zur Generalisierung und seine Reaktivität. Die Architektur des DPA-RL-Agenten ist modular aufgebaut. Die einzelnen Bestandteile des Systems sind austauschbar. Diese Möglichkeiten werden auch in den Untersuchungen ausgeschöpft. Unterschiedliche Realisierungen der abstrakten Aktionen werden untersucht, ohne dass gravierende Unterschiede festgestellt werden, so dass je nach Verfügbarkeit der Realisierungen der reaktiven Fähigkeiten jede Variante angewendet werden kann.
Innerhalb von zwei bis drei Episoden werden brauchbare Strategien erzielt. Die Untersuchungen für komplexe Szenarien werden direkt in der Realität durchgeführt.

9.4 Ausblick

Die heuristische Erweiterung ist in der vorliegenden Arbeit als Idee formuliert. Unterschiedliche Definitionen der heuristischen Funktionen, sowie die Anwendung in weiteren Problemen könnten große Vorteile aufdecken.
Die Möglichkeit zur Vorinitialisierung der Bewertungsfunktion mit vorhandenen Informationen bzw. die Möglichkeit die Informationen während der Ausführung der Mission für den Kraftfelder-Ansatz zu vervollständigen und mithilfe des SARSA-Algorithmus zu adaptieren wird in der vorliegenden Arbeit eingeführt. Weiterführende Untersuchungen in realen Problemstellungen werden aber nicht durchgeführt. Die vorgeschlagenen Ideen könnten für die Roboter-Navigation von Bedeutung sein und sollten weiterverfolgt werden.
Der Nachteil des Systems, sowohl im RL-Agenten als auch im DPA-RL-Agenten, liegt in den verwendeten Sensoren, die nur punktuelle Messungen in der Umgebung vornehmen können. Das Problem kann durch das am Lehrstuhl Intelligente Systeme entwickelte Radarsystem behoben werden, vergleiche [Zhigun, 2011].
Die entwickelte DPA-RL-Architektur ist modular und erlaubt weitere Aufsätze. Eine wichtige Annahme der entwickelten Struktur ist die sichere Feststellung des aktuellen Zustandes. Der aktuelle Zustand ist nicht immer sicher bekannt, kann aber mit Einbindung der Selbstlokalisierungsmethoden besser erfasst werden. Die Erzeugung weiterer Aktionen kann methodisch weiterentwickelt werden.
Folgende Punkte können verfolgt werden, um das System zu erweitern:

- Erweiterter Mechanismus zur Bestimmung der Zwischenziele, um den Aktionsraum dynamisch zu erweitern
- Bildung und Einbindung weiterer reaktiver Fähigkeiten, die ein noch komplexeres Gesamtverhalten des Agenten erlauben
- Entwicklung und Einbindung neuer Sensorik für den Roboter
- Prüfung der geschätzten Position und Erstellung einer Karte, um ein Matching durchzuführen und erlernte Strecken für neue Navigationsprobleme anzuwenden

Die abstrakte Aktion ”zum Ziel fahren” wird durch Vorgabe der Zielposition spezifiziert. Die Position des Ziels kann aber auch durch visuelle Merkmale ersetzt werden und mit Bildverarbeitungsalgorithmen aus Kamerabildern errechnet werden. Die Fähigkeit sich in Richtung Ziel zu bewegen, kann durch eine Regelung auf Grundlage der visuellen Merkmale ersetzt werden[48], diese Methode wird in Kapitel 6.4 beschrieben. Diese abstrakte Aktion kann auf höherer Abstraktionsebene realisiert werden. Zum Beispiel, wenn statt eines Punktes im Umgebungskoordinatensystem oder eines Objektes im Bild ein abstrakter Begriff wie ”Fahre zum Baum” genannt wird. In diesem Fall wird die Aufgabe auch einen Klassifikator benötigen. Als Klassifikator kann zum Beispiel der Ansatz der Support-Vektor-Maschinen genannt werden.

[48] Diese Regelungsart heißt auch Visual Servoing.

Anhang A

Anpassungsschritt für die verdeckte Schicht im RBF-Netzwerk

An dieser Stelle wird die Herleitung der im Abschnitt 3.4.5 eingeführten Lernregel für die Parameter in der verdeckten Schicht des RBF-Netzwerks durchgeführt. Das ”echte” RBF-Netzwerk wird dann für die Approximation der Q-Funktion angewendet.
Die aktuelle Fehlerfunktion im aktuellen Zustand s für die aktuelle Aktion a wird aus der aktuellen Abweichung zwischen der Messung $r + \gamma \cdot max_{a'} Q(\tau(s,a), a')$ und dem erwarteten Wert $Q(s,a)$ gebildet. Dieser Fehler ist auch als TD-Fehler bekannt.

$$\epsilon(s,a) = \frac{1}{2} \cdot (r + \max_{a'} \gamma Q(\tau(s,a), a') - Q(s,a))^2 = \frac{1}{2}\delta(s,a)^2 \tag{A.1}$$

Der neue Wert der Q-Funktion für das aktuelle Paar (s,a) soll so angepasst werden, dass sich der TD-Fehler verringert. Dieser Schritt wird numerisch mithilfe stochastischer Gradientenabstiegsverfahren berechnet:

$$\begin{aligned} Q^{neu}(s,a) &= Q^{alt}(s,a) - \alpha \cdot \frac{\partial \epsilon(s,a)}{\partial Q(s,a)} \\ &= Q^{alt}(s,a) + \alpha \cdot \underbrace{(r + \gamma \max_{a'} Q(\tau(s,a), a') - Q(s,a))}_{:=\delta(s,a)} \end{aligned} \tag{A.2}$$

In Gleichung (A.2) ist der gewöhnliche SARSA-Lernschritt für die Adaption der Q-Funktion dargestellt. Wenn der Zustandsraum nicht nur eine endliche Anzahl an Zuständen beinhaltet, wird die Q-Funktion durch eine lineare Funktion approximiert:

$$Q(s,a) = \sum_{i=1}^{N} \theta_a(i) \cdot w_s(i) = \sum_{i \in F(s)} \theta_a(i) w_s(i) \tag{A.3}$$

Wobei $w_s(i)$ für jedes $i \in F(s)$ die Gewichtung des Parameters $\theta_a(i)$ in der Summe für die Bildung des $Q(s,a)$-Wertes ist. $F(s)$ ist die Merkmalmenge für den Zustand s. Durch die Merkmalmenge wird der

Zustand s im diskretisierten Raum repräsentiert, siehe auch Kapitel 3.4.3 bzw. 3.4.4. Für das Coarse-Coding-Verfahren ist $w_s(i) = \frac{1}{|F(s)|}$, für radiale Basisfunktionen wird das Gewicht wie folgt definiert:

$$w_s(i) = \frac{\underbrace{\exp(-\frac{1}{2}(\mathbf{s}-\mu_\mathbf{i})^T\Sigma_i^{-1}(\mathbf{s}-\mu_\mathbf{i}))}_{:=\phi_s(i)}}{\sum_{j\in F(s)}\underbrace{\exp(-\frac{1}{2}(\mathbf{s}-\mu_\mathbf{j})^T\Sigma_k^{-1}(\mathbf{s}-\mu_\mathbf{j}))}_{=\phi_s(j)}} \tag{A.4}$$

Im Fall der linearen Approximation der Q-Funktion soll der $\theta_\mathbf{a}$-Parameter adaptiert werden. Für die Adaption des $\theta_\mathbf{a}$-Parameters wird auch das stochastische Gradientenabstiegsverfahren angewendet, bei dem der TD-Fehler (A.1) minimiert werden soll. Die Anpassung im Parametervektor erfolgt immer an den Stellen, durch die der aktuelle Zustand repräsentiert wird, also für jedes $i \in F(s)$. Sei $s' = \tau(s,a)$ und $a' \leftarrow \epsilon - gierig(Q(s',a))$ sowie $\delta_t(s,a)$ der aktuelle TD-Fehler, dann soll die Korrektur des Parametervektors wie folgt vorgenommen werden:

$$\begin{aligned}\theta_a^{neu}(i) &= \theta_a^{alt}(i) - \alpha\cdot(r+\gamma Q(s',a') - Q(s,a))\cdot\underbrace{\frac{-\partial Q(s,a)}{\partial\theta_a(i)}}_{:=w_s(i)}\\ &= \theta_a^{alt}(i) + \alpha\cdot\delta_t(s,a)\cdot w_s(i)\end{aligned} \tag{A.5}$$

Wenn in der Approximation der Bewertungsfunktion (A.3) die Parameter nicht nur in θ vorhanden sind, sondern die Merkmale im Merkmalvektor $w_s(i) = \frac{1}{Norm}\phi_s(i)$ auch verändert werden können, wird die Bewertungsfunktion durch eine nichtlineare Funktion approximiert. Wenn die Merkmalbasis als Menge von Gauß'schen Funktionen gewählt ist, wird die Q-Funktion durch ein RBF-Netzwerk approximiert. Der TD-Fehler kann verringert werden, wenn sowohl die Ausgabeschicht als auch die verdeckte Schicht des RBF-Netzwerks verändert wird. Die Lernregel für die Ausgabeschicht ist in Gleichung (A.5) gegeben. Hier wird die Herleitung für die Adaption des Parameters μ der verdeckten Schicht hergeleitet.
Sei $i \in F(s)$ und $\delta_t(s,a)$ der aktuelle TD-Fehler, dann wird die Lernregel für die Adaption von μ nach der Anwendung des stochastischen Gradientenabstiegsverfahrens wie folgt aussehen:

$$\mu_\mathbf{i}^\mathbf{neu} = \mu_\mathbf{i}^\mathbf{alt} - \alpha_\mu\nabla_{\mu_\mathbf{i}}\epsilon(s,a) \tag{A.6}$$

Die Dimension des Parameters $\mu_\mathbf{i}$ ist gleich der Dimension des Zustandes $\mathbf{s}$, $k = 1,\ldots K$. Die partielle Ableitung für die Koordinate k wird wie folgt definiert:

$$-\frac{\partial\epsilon(s,a)}{\partial\mu_i(k)} = (r+\gamma Q(s',a') - Q(s,a))\cdot\frac{\partial Q(s,a)}{\partial\mu_i(k)} \tag{A.7}$$

Die Ableitung der Q-Funktion nach Parameter $\mu(j)$ ist mit der Anwendung der Kettenregel in Gleichung (A.8) gegeben.

$$\begin{aligned}\frac{\partial Q(s,a)}{\partial\mu_i(k)} &\overset{(A.3)}{=} \frac{\partial\sum_{j\in F(s)} w_s(j)\theta_a(j)}{\partial\mu_i(k)}\\ &= \sum_{j\in F(s)}\theta_a(j)\cdot\frac{\partial w_s(j)}{\partial\mu_i(k)} = \theta_a(i)\cdot\frac{\partial w_s(i)}{\partial\mu_i(k)}\end{aligned} \tag{A.8}$$

Die folgenden Schritte werden für den Fall durchgeführt, in dem die Kovarianzmatrix (A.9) nur Einträge auf der Diagonalen hat. Es ist auch denkbar diesen Lernschritt auch für den allgemeinen Fall zu erweitern.

$$\Sigma_i^{-1} = \begin{pmatrix} \frac{1}{\sigma_i(1)^2} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_i(2)^2} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\sigma_i(3)^2} & \dots & 0 \\ & & \dots & & \\ 0 & 0 & 0 & \dots & \frac{1}{\sigma_i(K)^2} \end{pmatrix} \tag{A.9}$$

Zuerst wird ein Hilfsschritt (A.10) eingeführt. Dieser Schritt wird in Gleichung (A.11) eingesetzt, um die Ableitung $\frac{\partial w_s(i)}{\partial \mu_i(k)}$ zu bestimmen, wobei $w_s(i) = \frac{\phi_s(i)}{\sum_{j \in F(s)} \phi_s(j)}$.

$$\begin{aligned} \frac{\partial \phi_s(i)}{\partial \mu_i(k)} &= \frac{\partial \exp(-\frac{1}{2}(\mathbf{s} - \mu_\mathbf{i})^T \Sigma_i^{-1}(\mathbf{s} - \mu_\mathbf{i}))}{\partial \mu_i(k)} \\ &= \exp(-\frac{1}{2}(\mathbf{s} - \mu_\mathbf{i})^T \Sigma_i^{-1}(\mathbf{s} - \mu_\mathbf{i})) \cdot \frac{(s(k) - \mu_i(k))}{2 \cdot \sigma_i^2(k)} \\ &= \phi_s(i) \cdot \frac{(s(k) - \mu_i(k))}{\sigma_i^2(k)} \end{aligned} \tag{A.10}$$

$$\begin{aligned} \frac{\partial w_s(i)}{\partial \mu_i(k)} &= \frac{\frac{\partial \phi_s(i)}{\partial \mu_i(k)} \sum_{j \in F(s)} \phi_s(j) - \phi_s(i) \cdot \frac{\partial \sum_{j \in F(s)} \phi_s(j)}{\partial \mu_i(k)}}{(\sum_{j \in F(s)} \phi_s(j))^2} \\ &\overset{(A.10)}{=} \frac{\phi_s(i)}{\sum_{j \in F(s)} \phi_s(j)} \cdot \left[\frac{\frac{s(k) - \mu_i(k)}{\sigma_i^2(k)} \cdot \sum_{j \in F(s)} \phi_s(j) - \phi_s(j) \cdot \frac{s(k) - \mu_i(k)}{\sigma_i^2(k)}}{\sum_{j \in F(s)} \phi_s(j)} \right] \\ &= w_s(i) \cdot (1 - w_s(i)) \cdot \frac{s(k) - \mu_i(k)}{\sigma_i^2(k)} \end{aligned} \tag{A.11}$$

Wenn in Gleichung (A.6) die Gleichungen (A.7), (A.8), (A.11) eingesetzt werden, dann wird die Lernregel für den μ-Parameter $i \in F(s)$ wie folgt bestimmt:

$$\mu_\mathbf{i}^\mathbf{neu} = \mu_\mathbf{i}^\mathbf{alt} + \alpha_\mu \cdot \delta_t(s, a) \cdot \theta_a(i) \cdot w_s(i) \cdot (w_s(i) - 1) \cdot \Sigma_i^{-1}(\mathbf{s} - \mu_\mathbf{i}^\mathbf{alt}) \tag{A.12}$$

Der Lernschritt A.12 wird für die Anpassung der Position der jeweiligen RBF $\phi(i)$ aus der Merkmalmenge, also Parameter $\mu(i)$, angewendet. Die Ausdehnung der jeweiligen RBF $\phi(i)$, die durch den Parameter σ_i repräsentiert ist, wird nicht mithilfe des Gradientenabstiegsverfahrens berechnet, sondern in Abhängigkeit von der Verteilung der RBF definiert. Die neue Ausdehnung wird so bestimmt, dass jeder Zustand im Zustandsraum durch mindestens eine der angewendeten RBF repräsentiert wird. Dieser Schritt ist in Kapitel 3.4.5 ausführlich beschrieben.

Anhang B

Ablauf des SARSA*-Algorithmus im Gitterweltproblem

In diesem Abschnitt werden die Rechenschritte des SARSA-Algorithmus bzw. des SARSA*-Algorithmus im Gitterweltproblem für die Umgebung mit Weggabelung, siehe Abbildung 4.2(a), dargestellt, um eine bessere Nachvollziehbarkeit der eingeführten Erweiterungen zu gewährleisten. Der Diskontierungsfaktor wird aus Gründen der Übersichtlichkeit zu $\gamma = 1$ gesetzt. Das Belohnungsmodell ist wie folgt definiert. Jeder Schritt wird bestraft, also $r_t = -1$, außer wenn der Zielzustand (hier s_0) erreicht ist. In diesem Fall erhält der Agent den Wert Null. Es werden drei Rechenbeispiele für unterschiedliche heuristische Einflussparameter $k = \{0, 0.5, 1.0\}$ durchgeführt.
Die Aktualisierung der Bewertungsfunktion für den SARSA-Algorithmus wird mithilfe des in Gleichung (B.1) dargestellten Lernschrittes durchgeführt.

$$Q^{neu}(s,a) = Q^{alt}(s,a) + \alpha \cdot (r + \gamma \cdot max_a Q(\underbrace{\tilde{\tau}(s,a)}_{s_{nächst}}, a) - Q^{alt}(s,a)) \tag{B.1}$$

wobei $\tau(s,a)$ der Zustand ist, der vom Zustand s durch Ausführen der Aktion a erreicht werden kann, die nächste Aktion $a_{neu} \leftarrow max_a Q(\tau(s,a), a)$ wird mit der gierigen Strategie bestimmt.
Die Aktualisierung für den SARSA*-Algorithmus wird mithilfe der Gleichung (B.2) realisiert.

$$Q^{neu}(s,a) = Q^{alt}(s,a) + \alpha \cdot (r(s,a) + k \cdot \tilde{h}(s, s_{Ziel}) + \gamma \cdot max_a Q(\underbrace{\tau(s,a)}_{s_{nächst}}, a) - Q^{alt}(s,a)) \tag{B.2}$$

In diesem Beispiel wird die Q-Funktion in einer Iteration für alle Zustände $s_0, \ldots, s_{11}$ und Aktionen $a_0 = \uparrow, a_1 = \downarrow, a_2 = \leftarrow, a_3 = \rightarrow$ aktualisiert, siehe Algorithmus B.1. Die Lernschritte B.1 für den

Algorithmus B.1 Aktualisierung der Q-Tabelle mit dem SARSA*- bzw. SARSA-Algorithmus

$t = 0$, $Q = 0$
repeatfür jeden Lernschritt t
  for all $i = 0, \ldots 3$ **do** ▷ Für jede Aktion a_0, a_1, a_2, a_3
    for all $j = 0, \ldots 11$ **do** ▷ Für jeden Zustand $s_0, s_1, \ldots s_{11}$ des Zustandsraumes
      Aktualisiere $Q^t(s_i, a_j)$ gemäß Formel B.1 bzw. B.2
  $t = t + 1$
until Führe solange aus bis $\pi^*! = \pi \leftarrow Q^t$ ▷ Aktualisiere die Q-Funktion solange, bis die aus der Q-Funktion abgeleitete Strategie π mit der optimalen Strategie π^* übereinstimmt.

SARSA-Algorithmus ($k = 0$) sind in den Abbildungen B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9 dargestellt. Es werden 8 Iterationsschritte bis zum Erreichen der optimalen Strategie benötigt. Die resultierende optimale Strategie ist in Tabelle B.1 abgebildet.

Somit wird die optimale Strategie für das Standard-Verfahren nach 8 Iterationsschritten erreicht.

(a) $Q(a = \uparrow)$

0	|-1	-1	-1
0	|-1	-1	-1
-1	-1	-1	-1

(b) $Q(a = \downarrow)$

-1	|-1	-1	-1
-1	|-1	-1	-1
-1	-1	-1	-1

(c) $Q(a = \leftarrow)$

0	|-1	-1	-1
-1	|-1	-1	-1
-1	-1	-1	-1

(d) $Q(a = \rightarrow)$

0	|-1	-1	-1
-1	|-1	-1	-1
-1	-1	-1	-1

Abbildung B.1: $t = 0$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-2	-2	-2
0	|-2	-2	-2
-1	-2	-2	-2

(b) $Q(a = \downarrow)$

-1	|-2	-2	-2
-2	|-2	-2	-2
-2	-2	-2	-2

(c) $Q(a = \leftarrow)$

0	|-2	-2	-2
-1	|-2	-2	-2
-2	-2	-2	-2

(d) $Q(a = \rightarrow)$

0	|-2	-2	-2
-1	|-2	-2	-2
-2	-2	-2	-2

Abbildung B.2: $t = 1$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-3	-3	-3
0	|-3	-3	-3
-1	-3	-3	-3

(b) $Q(a = \downarrow)$

-1	|-3	-3	-3
-2	|-3	-3	-3
-2	-3	-3	-3

(c) $Q(a = \leftarrow)$

0	|-3	-3	-3
-1	|-3	-3	-3
-2	-2	-3	-3

(d) $Q(a = \rightarrow)$

0	|-3	-3	-3
-1	|-3	-3	-3
-3	-3	-3	-3

Abbildung B.3: $t = 2$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-4	-4	-4
0	|-4	-4	-4
-1	-3	-4	-4

(b) $Q(a = \downarrow)$

-1	|-4	-4	-4
-2	|-4	-4	-4
-2	-3	-4	-4

(c) $Q(a = \leftarrow)$

0	|-4	-4	-4
-1	|-4	-4	-4
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-4	-4	-4
-1	|-4	-4	-4
-3	-4	-4	-4

Abbildung B.4: $t = 3$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-5	-5	-5
0	|-5	-5	-5
-1	-3	-4	-5

(b) $Q(a = \downarrow)$

-1	|-5	-5	-5
-2	|-5	-5	-5
-2	-3	-4	-5

(c) $Q(a = \leftarrow)$

0	|-5	-5	-5
-1	|-5	-5	-5
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-5	-5	-5
-1	|-5	-5	-5
-3	-4	-5	-5

Abbildung B.5: $t = 4$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-6	-6	-6
0	|-6	-6	-6
-1	-3	-4	-6

(b) $Q(a = \downarrow)$

-1	|-6	-6	-6
-2	|-6	-6	-5
-2	-3	-4	-5

(c) $Q(a = \leftarrow)$

0	|-6	-6	-6
-1	|-6	-6	-6
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-6	-6	-6
-1	|-6	-6	-6
-3	-4	-5	-5

Abbildung B.6: $t = 5$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-7	-7	-7
0	|-7	-7	-7
-1	-3	-4	-6

(b) $Q(a = \downarrow)$

-1	|-7	-7	-6
-2	|-7	-7	-5
-2	-3	-4	-5

(c) $Q(a = \leftarrow)$

0	|-7	-7	-7
-1	|-7	-7	-7
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-7	-7	-7
-1	|-7	-6	-6
-3	-4	-5	-5

Abbildung B.7: $t = 6$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-8	-8	-7
0	|-8	-8	-7
-1	-3	-4	-6

(b) $Q(a = \downarrow)$

-1	|-8	-7	-6
-2	|-8	-7	-5
-2	-3	-4	-5

(c) $Q(a = \leftarrow)$

0	|-8	-8	-8
-1	|-8	-8	-7
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-8	-7	-7
-1	|-7	-6	-6
-3	-4	-5	-5

Abbildung B.8: $t = 7$, $k = 0$

(a) $Q(a = \uparrow)$

0	|-9	-8	-7
0	|-9	-8	-7
-1	-3	-4	-6

(b) $Q(a = \downarrow)$

-1	|-8	-7	-6
-2	|-8	-7	-5
-2	-3	-4	-5

(c) $Q(a = \leftarrow)$

0	|-9	-9	-8
-1	|-8	-8	-7
-2	-2	-3	-4

(d) $Q(a = \rightarrow)$

0	|-8	-7	-7
-1	|-7	-6	-6
-3	-4	-5	-5

Abbildung B.9: $t = 8$, $k = 0$

Tabelle B.1: Optimale Strategie für das Standard-Verfahren. $k = 0$, $\gamma = 1$

$\uparrow \leftarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow \rightarrow$	$\downarrow$
$\uparrow$	$\underline{\rightarrow}$	$\underline{\rightarrow}$	$\downarrow$
$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

Zwei Beispiele, ein positives und ein negatives, wie sich der heuristische Einfluss auswirken kann, werden erläutert. Die positive Auswirkung des heuristischen Einflusses wird für den Parametersatz ($k = 0.5$ und $\gamma = 1.0$) erzielt. Der negative Einfluss des heuristischen Einflusses im SARSA*-Algorithmus wird mit dem Parametersatz ($k = 1.0$, $\gamma = 1.0$) erzielt.
Die aktuelle Strategie wird aus der aktuellen Q-Tabelle bestimmt, indem die gierige Wahl der Aktion aus der Q-Tabelle getroffen wird, so dass für jeden Zustand die Aktion ausgewählt wird, die den maximalen Gewinn verspricht.
In den Abbildungen B.10, B.11, B.12, B.13, B.14, sind bis zum Erreichen der optimalen Strategie, die in der Tabelle B.2 zu finden ist, 4 Iterationsschritte dargestellt. Dabei ist zu bemerken, dass sich die Strategie in Tabelle B.2 von der Strategie in Tabelle B.1 in den Zuständen s_1 und s_2 unterscheidet. In diesen Zuständen entsteht keine Wahl nach rechts oder nach unten zu gehen, sondern es wird nur die Möglichkeit nach rechts zu gehen als die optimale Aktion präsentiert.

(a) $Q(a = \uparrow)$

0	|-0.86	-0.72	-0.58
0.13	|-0.8	-0.68	-0.56
-0.58	-0.68	-0.6	-0.5

(b) $Q(a = \downarrow)$

-0.86	|-0.86	-0.72	-0.58
-0.86	|-0.8	-0.68	-0.56
-0.72	-0.68	-0.6	-0.5

(c) $Q(a = \leftarrow)$

0	|-0.86	-0.72	-0.58
-0.72	|-0.8	-0.68	-0.56
-0.72	-0.68	-0.6	-0.5

(d) $Q(a = \rightarrow)$

0	|-0.86	-0.72	-0.58
-0.72	|-0.8	-0.68	-0.56
-0.72	-0.68	-0.6	-0.5

Abbildung B.10: $t = 0$, $k = 0.5$

(a) $Q(a = \uparrow)$

0	|-1.72	-1.44	-1.16
0.13	|-1.66	-1.41	-1.14
-0.58	-1.37	-1.21	-1.06

(b) $Q(a = \downarrow)$

-0.86	|-1.66	-1.41	-1.14
-1.44	|-1.6	-1.37	-1.06
-1.3	-1.37	-1.21	-1

(c) $Q(a = \leftarrow)$

0	|-1.72	-1.58	-1.3
-0.72	|-1.6	-1.49	-1.25
-1.3	-1.27	-1.29	-1.1

(d) $Q(a = \rightarrow)$

0	|-1.58	-1.3	-1.16
-0.72	|-1.49	-1.25	-1.12
-1.41	-1.29	-1.1	-1

Abbildung B.11: $t = 1$, $k = 0.5$

(a) $Q(a = \uparrow)$

0	|-2.44	-2.02	-1.72
0.13	|-2.38	-1.99	-1.7
-0.58	-1.96	-1.71	-1.56

(b) $Q(a = \downarrow)$

-0.86	|-2.35	-1.97	-1.64
-1.44	|-2.29	-1.94	-1.56
-1.3	-1.96	-1.71	-1.5

(c) $Q(a = \leftarrow)$

0	|-2.44	-2.3	-1.89
-0.72	|-2.29	-2.18	-1.81
-1.3	-1.27	-1.88	-1.6

(d) $Q(a = \rightarrow)$

0	|-2.16	-1.89	-1.75
-0.72	|-2.05	-1.81	-1.68
-1.99	-1.79	-1.6	-1.5

Abbildung B.12: $t = 2$, $k = 0.5$

Die optimale Strategie, siehe Tabelle B.2 wird nach 4 Iterationen erreicht. Ein Beispiel für unkorrekt bestimmte Parameter für den heuristischen Einfluss wird hier dargestellt. Es sind nur zwei ausgewählte

(a) $Q(a = \uparrow)$

0	|-3.02	-2.61	-2.22
0.13	|-2.97	-2.58	-2.2
-0.58	-1.96	-2.21	-2.06

(b) $Q(a = \downarrow)$

-0.86	|-2.91	-2.53	-2.14
-1.44	|-2.85	-2.5	-2.06
-1.3	-1.96	-2.21	-2

(c) $Q(a = \leftarrow)$

0	|-3.02	-2.89	-2.47
-0.72	|-2.85	-2.74	-2.37
-1.3	-1.27	-1.88	-2.1

(d) $Q(a = \rightarrow)$

0	|-2.75	-2.47	-2.33
-0.72	|-2.61	-2.37	-2.24
-1.99	-2.29	-2.1	-2

Abbildung B.13: $t = 3$, $k = 0.5$

(a) $Q(a = \uparrow)$

0	|-3.61	-3.19	-2.72
0.13	|-3.55	-3.16	-2.7
-0.58	-1.96	-2.48	-2.56

(b) $Q(a = \downarrow)$

-0.86	|-3.47	-3.09	-2.64
-1.44	|-3.42	-3.06	-2.56
-1.3	-1.96	-2.48	-2.5

(c) $Q(a = \leftarrow)$

0	|-3.61	-3.47	-3.05
-0.72	|-3.42	-3.3	-2.93
-1.3	-1.27	-1.88	-2.38

(d) $Q(a = \rightarrow)$

0	|-3.33	-3.05	-2.91
-0.72	|-3.17	-2.93	-2.8
-1.99	-2.57	-2.6	-2.5

Abbildung B.14: $t = 4$, $k = 0.5$

Tabelle B.2: Optimale Strategie für SARSA*-Verfahren. $k = 0.5$, $\gamma = 1$

$\uparrow \leftarrow \rightarrow$	$\vert\rightarrow$	$\rightarrow$	$\downarrow$
$\uparrow$	$\vert\underline{\rightarrow}$	$\underline{\rightarrow}$	$\downarrow$
$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

Iterationen $t = 0$ und $t = 499$ dargestellt, siehe Abbildung B.15, B.16, sowie eine nicht optimale Strategie, siehe Tabelle B.3. In dieser Tabelle kann die Entstehung der in Kapitel 4.2.3 beschriebenen Zyklen beobachtet werden. Nach Meinung des Agenten ist es sinnvoll sich im Zustand s_{11} aufzuhalten anstatt zum Ziel zu navigieren. Die gewählte Parametrisierung ($k = 1.0, \gamma = 1.0$) verletzt das zulässige Intervall, siehe Gleichung (4.20).

In Tabelle B.4 ist ein weiteres Beispiel für die Konvergenz zur nicht-optimalen Strategie dargestellt. In

(a) $Q(a = \uparrow)$

0	|-0.72	-0.44	-0.16
0.27	|-0.6	-0.37	-0.12
-0.16	-0.37	-0.21	0

(b) $Q(a = \downarrow)$

-0.72	|-0.72	-0.44	-0.16
-0.72	|-0.6	-0.37	-0.12
-0.44	-0.37	-0.21	0

(c) $Q(a = \leftarrow)$

0	|-0.72	-0.44	-0.16
-0.44	|-0.6	-0.37	-0.12
-0.44	-0.37	-0.21	0

(d) $Q(a = \rightarrow)$

0	|-0.72	-0.44	-0.16
-0.44	|-0.6	-0.37	-0.12
-0.44	-0.37	-0.21	0

Abbildung B.15: $t = 0$, $k = 1$

(a) $Q(a = \uparrow)$

0	|-2.18	-1.18	-0.45
0.27	|-2.06	-1.11	-0.41
-0.16	-0.92	-0.43	-0.12

(b) $Q(a = \downarrow)$

-0.72	|-1.83	-0.94	-0.29
-0.89	|-1.71	-0.88	-0.12
-0.61	-0.92	-0.43	0

(c) $Q(a = \leftarrow)$

0	|-2.18	-1.9	-0.9
-0.44	|-1.71	-1.49	-0.62
-0.61	-0.54	-0.76	-0.21

(d) $Q(a = \rightarrow)$

0	|-1.45	-0.73	-0.45
-0.44	|-1.11	-0.5	-0.24
-0.99	-0.59	-0.21	0

Abbildung B.16: $t = 499$, $k = 1$

Tabelle B.3: Nicht-optimale Strategie, erzielt mit dem SARSA*-Algorithmus und einem heuristischen Einfluss von $k = 1$, $\gamma = 1.0$. Der Zyklus entsteht im Zustand s_{11}, siehe Zustand unten rechts.

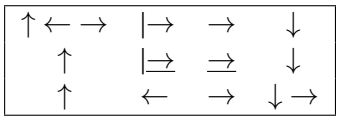
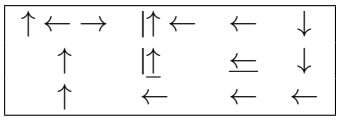


Tabelle B.4: Nicht-optimale Strategie, erzielt mit dem SARSA*-Algorithmus und einem heuristischen Einfluss von $k = -15.0$, $\gamma = 0.9$. Der Zyklus entsteht im Zustand s_1.



diesem Beispiel ist der Diskontierungsfaktor auf $\gamma = 0.9$ gesetzt und der heuristische Einflussparameter auf $k = -15$ gesetzt. Der Zyklus hat sich an einem anderen Zustand s_1 gebildet als in Tabelle B.3.

Anhang C

3D-Mountain-Agent-Problem

C.1 Problemstellung

Das an dieser Stelle eingeführte Beispiel für ein 3D-Mountain-Agent-Problem wird in [Noglik et al., 2009] präsentiert. Dieses Beispiel ist durch die Erweiterung des Mountain-Car-Problems (MCP) um zwei Koordinaten entstanden. Der Raum, in dem sich der Agent bewegen kann, hat drei Dimensionen, wobei die Höhe des Agenten latent im System bleibt und seine Position aus zwei Dimensionen besteht. Die Landschaft ist in Abbildung C.1 dargestellt. Der Zielzustand befindet sich in der Mitte der Landschaft auf einem Berg und ist auf dem Bild durch blaue Punkte dargestellt, $(0.523, 0.523)$. Der Startzustand befindet sich in einem Tal und ist durch den grünen Ball dargestellt. Diese Position hat die Koordinaten $(-0.523, -0.523)$. Die Geschwindigkeit beim Start vom Startpunkt ist Null. Der Geschwindigkeitsvektor des Agenten ist in dieser Aufgabenstellung zweidimensional. Somit hat der Zustandsraum vier Dimensionen: $(x, \dot{x}, y, \dot{y})$.

$$s := ((x, y), (\dot{x}, \dot{y})) \in ([-1.2, 2.2] \times [-1.2, 2.2]) \times ([-0.05, 0.05] \times [-0.05, 0.05])\}$$

Der Aktionsraum enthält 9 mögliche Aktionen $A(s) = \{(0,0), (+/-1, 0), (0, +/-1), (+/-1 \cdot \frac{1}{2}, +/-1 \cdot \frac{1}{2})$ für jeden Zustand s. Die Aufgabenstellung wird als 3D-Mountain-Agent-Problem (3D-MAP) bezeichnet und nicht als 3D-Mountain-Car-Agent, da sich der Agent, der die vorgeschlagenen Aktionen ausführen muss, sowohl vorwärts als auch seitlich bewegen kann, also ein holonomes System sein soll. Im Gegensatz dazu können handelsübliche Kraftfahrzeuge (engl.: cars) nicht seitlich fahren und stellen damit nicht-holonome Systeme dar.
Als Diskretisierungsmethode für den vierdimensionalen Zustandsraum wird Tile-Coding gewählt. Dabei wird ein Tiling verwendet, das in

$$\underbrace{18}_{\text{für x-Anteil}} \times \underbrace{18}_{\text{für y-Anteil}} \times \underbrace{9}_{\text{für } \dot{x}\text{-Anteil}} \times \underbrace{9}_{\text{für } \dot{y}\text{-Anteil}}$$

Teile zerlegt ist.
Das Gleichungssystem der Zustandsänderung in Abhängigkeit vom Steuersignal (a_x, a_y) sieht den Gleichungen (2.2, 2.1) ähnlich, es soll nur für die vier Komponenten $(x, y, \dot{x}, \dot{y})$ berechnet werden. Die Dimensionalität dieser Aufgabenstellung ist viel höher, der Parametervektor hat die Dimensionalität: $\underbrace{18 \times 18 \times 9 \times 9}_{\text{für Zustandsraum}} \times \underbrace{9}_{\text{für Aktionsraum}}$, im Vergleich zu $9 \times 9 \times 3$ für das Standard-MCP. Außerdem werden in dieser Problemstellung die Navigationsfähigkeiten des Agenten hervorgehoben, da das Ziel nicht nur auf dem Berg liegt, sondern auch noch gesucht werden muss.

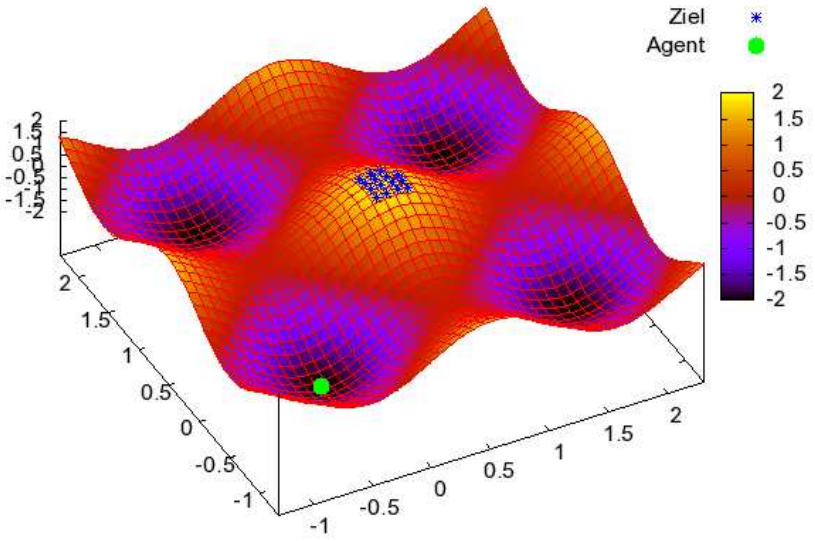


Abbildung C.1: 3D-Mountain-Agent-Problem

C.2 3D-Mountain-Agent-Problem für den SARSA*-Algorithmus

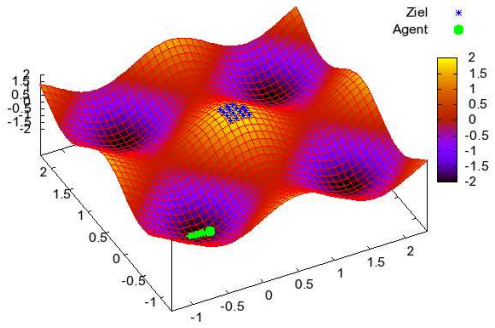
Hier wurden nur wenige Parameterwerte untersucht, da die durchgeführten Experimente zeitintensiv sind. In Tabelle C.1 sind die Ergebnisse aufgelistet. Die Anwendung der heuristischen Funktion erzielt

Tabelle C.1: Prozentualer Unterschied zur Standard-Methode bei Anwendung der heuristischen Funktion 4.33 im SARSA-Algorithmus im 3D-MAP, Parametrisierung: $\epsilon = 0.0, \alpha = 0.5, \lambda = 0.92, \gamma = 1.0$

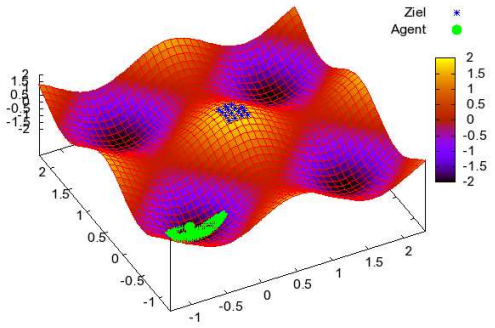
k	LP	t-Test LP
0.5	-1.17%	0
0	0%	0
-0.5	2.75%	1

wieder eine Verbesserung. Die Verbesserung ist zwar signifikant, siehe dritte Zeile der Tabelle C.1, allerdings ist sie mit 2.75% im Vergleich zur Standard-Methode relativ klein.
In Abbildung C.2 und Abbildung C.3 ist der Verlauf der erzielten Strategie nach der 999. Episode dargestellt. Der Lernprozess basiert auf dem SARSA*-Algorithmus.
Das Ziel-Areal für den Agenten ist durch blaue Punkte gekennzeichnet. Die vom Agenten verfolgte Trajektorie ist mit grünen Kreuzwolken in jedem Bild gekennzeichnet.
Zuerst soll der Agent aus dem Tal herausfahren, in dem er sich befindet. Über 500 Schritte werden dafür benötigt, siehe Abbildungen C.2(a), C.2(b), C.2(c). Wenn der Agent ein gewisses Niveau erreicht hat, fängt er an Kreise um das Tal zu drehen, siehe Abbildung C.2(d). Der Agent befindet sich schon in unmittelbarer Nähe des Ziels. Innerhalb der nächsten 165 Schritte erreicht er dann sein vorgegebenes Ziel, siehe Abbildung C.3.
Im $800.$ Schritt hat der Agent das Ziel fast erreicht, siehe Abbildung C.3(a). Ihm fehlte nur wenig

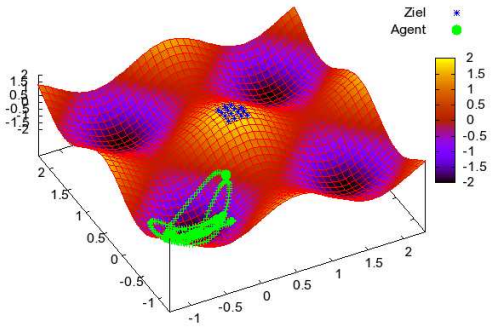
Schwung, um das Ziel-Areal zu erreichen. Ab dem ungefähr 820. Schritt fängt der Agent den Versuch an, der mit Erfolg endet. Der Agent fängt seinen letzten Versuch wieder von der Ecke an und holt den benötigten Schwung von den tieferen Punkten, versucht aber auf dem erreichten Niveau zu bleiben. Hier fährt der Agent über einen noch tieferen Punkt des Tals, holt den benötigten Schwung und erreicht das Ziel innerhalb der nächsten 50 Schritte, siehe Abbildungen C.3(b), C.3(c), C.3(c).
Eine Erklärung für die geringe Verbesserung des SARSA∗-Algorithmus im Vergleich zum Standard-SARSA-Algorithmus im 3D-MAP könnte die Definition der heuristischen Funktion sein. Die Funktion wurde zielpunktbasiert definiert, d.h. die Zustände, die nahe am Ziel-Areal liegen, bringen dem Agenten eine höhere Belohnung bzw. geringere Bestrafung ein. Eine heuristische Funktion, die auf dem Startpunkt basiert, würde dem Agenten vielleicht einen größeren Nutzen bringen. In diesem Fall werden die Zustände umso höher belohnt, je weiter weg sie sich vom Tiefpunkt im Tal befinden. Der Agent würde sich also erst einmal darauf konzentrieren aus dem Tal herauszukommen.



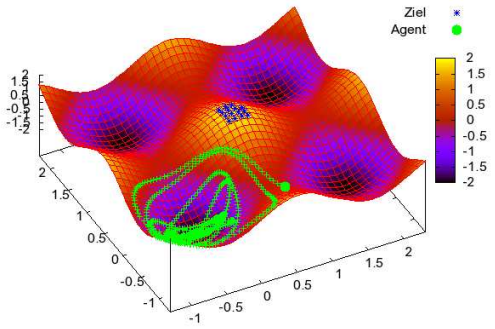
(a) Schritt 100



(b) Schritt 300

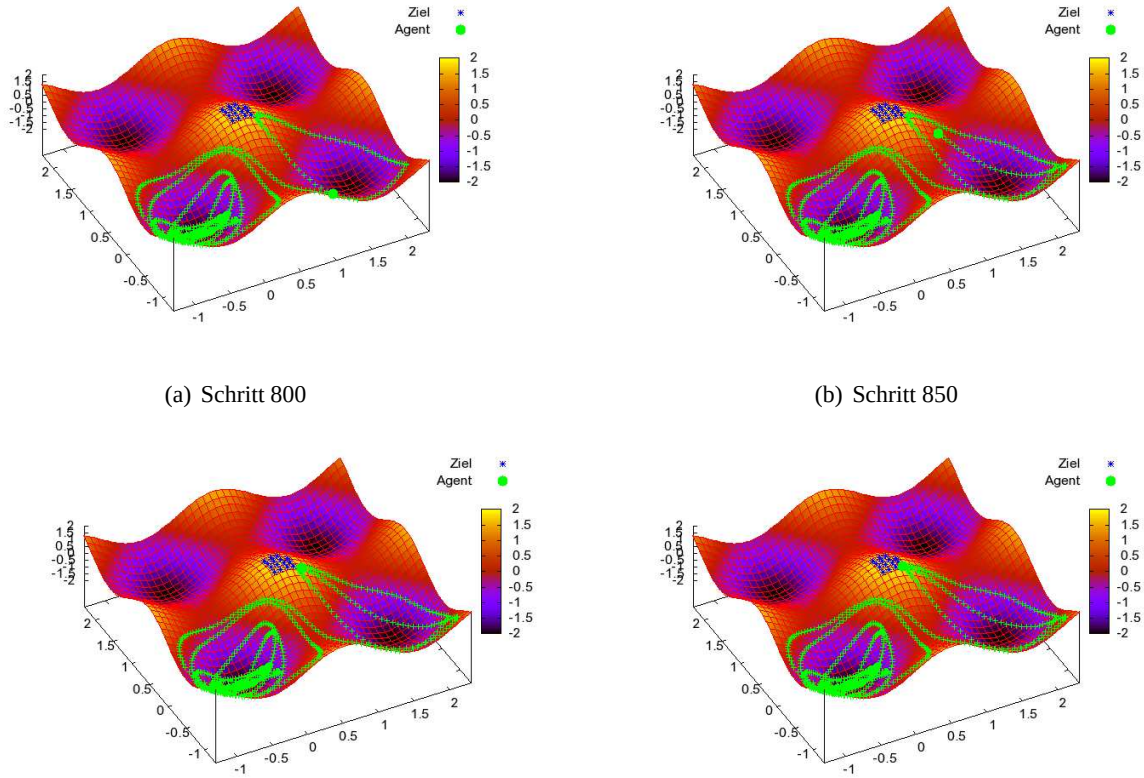


(c) Schritt 500



(d) Schritt 700

Abbildung C.2: Lernprozess für das 3D-Mountain-Agent-Problem, 100., 300., 500. und 700. Schritt der 999. Episode mit einem heuristischen Einfluss von -0.5



(a) Schritt 800

(b) Schritt 850

(c) Schritt 860

(d) Schritt 865

Abbildung C.3: Lernprozess für das 3D-Mountain-Agent-Problem, 800., 850., 860., und 865. Schritt der 999. Episode mit einem heuristischen Einfluss von -0.5

Literaturverzeichnis

[Albus, 1971] ALBUS, J.: *A Theory of Cerebellar Function*. In: *Mathematical Biosciences*, 10 (1971), Seiten 25–61

[Albus, 2002] ALBUS, J.: *4D/RCS: A Reference Model Architecture for Intelligent Unmanned Ground Vehicles*. In: *In Proceedings of SPIE Aerosense Conference*, 2002, Seiten 1–5

[Arkin, 1998] ARKIN, R.: *Behavior-Based Robotics*. 3. Auflage. Cambridge, MA : MIT Press, 1998

[Arras & Siegwart, 2000] ARRAS, K.; SIEGWART, R.: *Acht häufig gestellte Fragen an die Mobilrobotik*. In: *SGA-Bulletin*, 14 (2000), Seiten 1–5

[Baird et al., 1999] BAIRD, L.; FAHLMAN, S.; KAELBLING, L.: *Reinforcement Learning Through Gradient Descent*. 1999

[Berger, 2006] BERGER, R.: *Die Doppelpass-Architektur Verhaltenssteuerung autonomer Agenten in dynamischen Umgebungen*. – Diplomarbeit. Humboldt Universität, Berlin, Germany, März 2006

[Berger et al., 2006] BERGER, R.; HEIN, D.; BURKHARD, H.: *AT Humboldt 2006 & AT Humboldt 3D - Team Description*. In: *RoboCup 2006 - Proceedings of the International Symposium*, 2006 (LNAI)

[Bianchi et al., 2008] BIANCHI, R.; RIBEIRO, C.; COSTA, A.: *Accelerating autonomous learning by using heuristic selection of actions*. In: *Journal of Heuristics*, 14 (2008), Nr. 2, Seiten 135–168

[Boyan, 2002] BOYAN, J.: *Technical Update: Least-Squares Temporal Difference Learning*. In: *Machine Learning*, 49 (2002), Nr. 2-3, Seiten 233–246

[Bradtke et al., 1996] BRADTKE, S.; BARTO, A.; KAELBLING, P.: *Linear least-squares algorithms for temporal difference learning*. In: *Machine Learning* Bd. 22, 1996, Seiten 22–33

[Bratman, 1987] BRATMAN, M.: *Intentions, Plans, and Practical Reason*. University Press, Harvard, 1987

[Bratman et al., 1988] BRATMAN, M.; ISRAEL, D.; POLLACK, M.: *Plans and Resource-Bounded Practical Reasoning*. In: *Computational Intelligence*, 4 (1988), Nr. 4, Seiten 349–355

[Buckland, 2002] BUCKLAND, M.: *Paket Windows NEAT*. 2002. – URL `http://nn.cs.utexas.edu/?windowsneat`

[Burkhard et al., 2002] BURKHARD, H.; BACH, J.; BERGER, R.; BRUNSWIECK, B.; GOLLIN, M.: *Mental Models for Robot Control*. In: *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, 2002

[Cocora et al., 2006] COCORA, A.; KERSTING, K.; PLAGEMANN, C.; BURGARD, W.; DE RAEDT, L.: *Learning Relational Navigation Policies*. In: *Kuenstliche Intelligenz: Special Issue on 'Lernen und Selbstorganisation von Verhalten'*, 06 (2006), Nr. 3, Seiten 12—18

[Deutsch, 2004] DEUTSCH, C.: *Pfadplanung für einen autonomen mobilen Roboter*. – Diplomarbeit. Institut für Regelungs- und Automatisierungstechnik, Technische Universität Graz, Austria, April 2004

[Dietterich, 2000a] DIETTERICH, T.: *Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition*. In: *Journal of Artificial Intelligence Research*, 13 (2000), Seiten 227–303

[Dietterich, 2000b] DIETTERICH, T.: *An Overview of MAXQ Hierarchical Reinforcement Learning*. In: *Lecture Notes in Computer Science* Bd. 1864 / 2000, Springer, 2000, Seiten 26 – 44

[Donnart & Meyer, 1995] DONNART, J.; MEYER, J.: *Learning Reactive and Planning Rules in a Motivationally Autonomous Animat*. In: *IEEE Transactions on Systems, Man, and Cybernetics, part B: Cybernetics*, 26 (1995), Seiten 381–395

[Dorigo & Colombetti, 1998] DORIGO, M.; COLOMBETTI, M.: *Robot Shaping: An Experiment in Behavior Engineering (Intelligent Robotics and Autonomous Agents)*. 1. Auflage. MIT Press, Massachusetts, 1998

[Efimenko, 2011] EFIMENKO, M.: *Erweiterung und Evaluierung DPA-RL Architektur*. – Master Thesis. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, August 2011

[Eickhoff, 2009] EICKHOFF, B.: *Generalisierung bei der Roboterlokalisierung mit Partikel-Filter*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, Mai 2009

[EvolutionRobotics, 2004] EVOLUTIONROBOTICS: *Manual, ERSP Scorpion, Application Development Robot*. 2004

[Fisher, 1987] FISHER, D.: *Knowledge acquisition via incremental conceptual clustering*. In: *Machine Learning*, 1987, Seiten 139–172

[Georgeff et al., 1999] GEORGEFF, M.; PELL, B.; POLLACK, M.; TAMBE, M.; WOOLDRIDGE, M.: *The Belief-Desire-Intention Model of Agency*. In: *Intelligent Agents V: Agents Theories, Architectures, and Languages*. Berlin, Heidelberg, 1999, Seiten 1–10

[Gerdes et al., 2004] GERDES, I.; KLAWON, F.; KRUSE, R.: *Evolutionäre Algorithmen*. 1. Auflage. Wiesbaden : Vieweg Verlag, Juli 2004

[Hamdi, 1999] HAMDI, M.: *Entwurf adaptiver lernender Roboter*. – Dissertation. Georg-August-Universität, Göttingen, Germany, 1999

[Hart et al., 1968] HART, P.; NILSSON, N.; RAPHAEL, B.: *A formal basis for the heuristic determination of minimum cost paths*. In: *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4 (1968), Nr. 2, Seiten 100–107

[Hartung, 1985] HARTUNG, J.: *Statistik : Lehr- und Handbuch der angewandten Statistik*. 3. Auflage. München : Oldenbourg Verlag, 1985

[Heidrich-Meisner et al., 2007] HEIDRICH-MEISNER, V.; LAUER, M.; IGEL, C.; RIEDMILLER, M.: *Reinforcement Learning in a Nutshell*. In: *15th European Symposium on Artificial Neural Networks*, 2007, Seiten 277–288

[Auf dem Hövel, 2002] HÖVEL, J. Auf dem: *Abenteuer Künstliche Intelligenz: Auf der Suche nach dem Geist in der Maschine*. Discorsi Verlag, Hamburg, 2002

[Hu & Wellman, 1998] HU, J.; WELLMAN, M.: *Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm*. 1998

[Hutchinson et al., 1996] HUTCHINSON, S.; HAGER, G.; CORKE, P.: *A tutorial on visual servo control*. In: *Robotics and Automation, IEEE Transactions on*, 12 (1996), Seiten 651 – 670

[Igel & Sendhoff, 2005] IGEL, C.; SENDHOFF, B.: *Synergies between Evolutionary and Neural Computation*. In: *European Symposium on Artificial Neural Networks, Bruges*, 2005, Seiten 241–252

[Jung, 2007] JUNG, T.: *Reinforcement Lernen mit Regularisierungsnetzwerken*. – Dissertation. Physik, Mathematik und Informatik, Johannes Gutenberg-Universität, November 2007

[Jung, 2008] JUNG, T.: *Reinforcement Lernen mit Regularisierungsnetzen*. In: *Ausgezeichnete Informatikdissertationen 2007*. Bonn : GI-Edition Lecture Notes in Informatics (LNI), Koellen Verlag, 2008

[Kassahun, 2006] KASSAHUN, Y.: *Towards a Unified Approach to Learning and Adaptation*. – Dissertation. der Technischen Fakultät der Christian-Albrechts-Universität, Kiel, 2006

[Khatib, 1986] KHATIB, O.: *Real-time obstacle avoidance for manipulators and mobile robots*. In: *International Journal of Robotic Research*, 5 (1986), Nr. 1, Seiten 90–98

[Koenig & Simmons, 1998] KOENIG, S.; SIMMONS, R.: *Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models*. In: *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, MIT Press, 1998, Seiten 91–122

[Kohonen, 1982] KOHONEN, T.: *Self-organized formation of topologically correct feature maps*. In: *Biological Cybernetics*, 43 (1982), Nr. 1, Seiten 59–69

[Köpsel et al., 2007] KÖPSEL, T.; NOGLIK, A.; PAULI, J.: *Evolutionäre Algorithmen zur Topologieentwicklung von Neuronalen Netzen für die Roboter-Navigation im praktischen Einsatz*. In: K. BERNS, T. (Hrsg.): *Autonome Mobile Systeme 2007*. Berlin : Springer-Verlag, 2007 (Informatik aktuell), Seiten 145–151

[Köpsel, 2007] KÖPSEL, T.: *Evolutionäre Algorithmen zur Topologieentwicklung von Neuronalen Netzen für die Roboter-Navigation im praktischen Einsatz*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, Juni 2007

[Langley, 2001] LANGLEY, P.: *The Computational Support of Scientific Discovery*. In: *Machine Learning and Its Applications*, 2001, Seiten 230–248

[Latombe, 1991] LATOMBE, J.: *Robot Motion Planning*. 1 st. Kluwer Academic Publishers, Norwell, USA, 1991

[Leonard & Durrant-Whyte, 1991] LEONARD, J.; DURRANT-WHYTE, H.: *Simultaneous map building and localisation for an autonomous mobile robot*. In: *Workshop on Intelligent Robots and Systems (IROS)*. Osaka, 1991, Seiten 1442–1447

[Lisowski, 2009] LISOWSKI, C.: *Robuste 3D-Objektklassifikation mit lokalen Strukturen und Support-Vektor-Maschinen*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, Februar 2009

[Mai, 2010] MAI, A.: *Reinforcment Lernen zur adaptiven Verhaltenssteuerung intelligenter Agenten in der Roboternavigation*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, Juli 2010

[Mazur, 2006] MAZUR, J.: *Lernen und Verhalten*. 6. Auflage. Pearson Studium, München, 2006

[Mellmann, 2011] MELLMANN, H.: *Ein anderes Modell der Welt: Alternative Methoden zur Lokalisierung Mobiler Roboter*. – Diplomarbeit. Lehrstuhl für Künstliche Intelligenz, Humboldt-Universität zu Berlin, Januar 2011

[Melo et al., 2008] MELO, F.; MEYN, S.; RIBEIRO, M.: *An analysis of reinforcement learning with function approximation*. In: *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008

[Menache, 2003] MENACHE, I.: *Basis Function Adaptation in Temporal Difference Reinforcement Learning*. – Technischer Bericht. Department of Electrical Engineering, Technion, Israel Institute of Technology, Israel, 2003

[Menache et al., 2005] MENACHE, I.; MANNOR, S.; SHIMKIN, N.: *Basis function adaptation in temporal difference reinforcement learning*. In: *Annals of Operations Research*, 134 (2005), Seiten 215–238

[Murphy, 2000] MURPHY, R.: *Introduction to AI Robotics*. 1. Auflage. Cambridge, MA : MIT Press, 2000

[Noglik et al., 2009] NOGLIK, A.; MÜLLER, M.; PAULI, J.: *Application of a Heuristic Function in Reinforcement Learning*. In: *Hybrid Control for Autonomous Systems Integrating Learning, Deliberation, and Reactive Control*, 2009, Seiten 41–48

[Noglik & Pauli, 2009] NOGLIK, A.; PAULI, J.: *Combination of Reinforcement Learning and Neural Networks*. In: *2nd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*, 2009, Seiten 9–17

[Papierok, 2007] PAPIEROK, S.: *Reinforcement Learning in realen Umgebungen unter Verwendung von radialen Basisfunktionen*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, Dezember 2007

[Papierok et al., 2008] PAPIEROK, S.; NOGLIK, A.; PAULI, J.: *Application of Reinforcement Learning in a Real Environment using RBF Networks*. In: *Proceedings of the International Workshop on Evolutionary Learning for Autonomous Robot Systems*, Juli 2008, Seiten 17–22

[Parr, 1998] PARR, R.: *Hierarchical Control and Learning for Markov Decision Processes*. – Dissertation. University of California, Berkeley, 1998

[Perkins & Precup, 2002] PERKINS, T.; PRECUP, D.: *A Convergent Form of Approximate Policy Iteration*. In: *NIPS*, 2002, Seiten 1595–1602

[Pfeifer & Bongard, 2006] PFEIFER, R.; BONGARD, J.: *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, 2006

[Poggio & Girosi, 1989] POGGIO, T.; GIROSI, F.: *A Theory of Networks for Approximation and Learning*. In: *Laboratory, Massachusetts Institute of Technology*, 1140 (1989)

[Poole et al., 1998] POOLE, D.; MACKWORTH, A.; GOEBEL, R.: *Computational Intelligence: A Logical Approach*. New York : Oxford University Press, January 1998

[Redlich & Henze, 2007] REDLICH, J.; HENZE, C.: *Jahresbericht 2007*. 2007

[Röttger, 2009] RÖTTGER, M.: *Reinforcement-Learning für kontinuierliche Zustands- und Aktionsräume unter Berücksichtigung der wissenschaftlichen Informationsverarbeitung*. Diplomarbeit. Freiburg, Fakultät für Mathematik und Physik, der Albert-Ludwigs-Universität, 2009

[Rummery & Niranjan, 1994] RUMMERY, G.; NIRANJAN, M.: *On-Line Q-Learning Using Connectionist Systems* / Cambridge University Engineering Department. 1994. – Forschungsbericht

[Russell & Norvig, 2004] RUSSELL, S.; NORVIG, P.: *Künstliche Intelligenz. Ein moderner Ansatz*. Pearson Studium, 2004

[Ryan, 2002] RYAN, M.: *Using abstract models of behaviours to automatically generate reinforcement learning hierarchies*. In: *In Proceedings of The 19th International Conference on Machine Learning*, Kaufmann, M., 2002, Seiten 522–529

[Singh et al., 2000] SINGH, S.; JAAKKOLA, T.; LITTMAN, M.; SZEPESVÁRI, C.: *Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms*. In: *Machine Learning*, 38 (2000), Nr. 3, Seiten 287–308

[Singh & Sutton, 1996] SINGH, S.; SUTTON, R.: *Reinforcement Learning with Replacing Eligibility Traces*. In: *Machine Learning*, 22 (1996), Seiten 123–158

[Sluzhivoy, 2008] SLUZHIVOY, A.: *Effiziente Methoden der Objekterkennung mit Support Vector Machines*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, März 2008

[Sluzhivoy et al., 2008] SLUZHIVOY, A.; PAULI, J.; RÖLKE, V.; NOGLIK, A.: *Improving Runtime Complexity of Multi-class Support Vector Machines*. In: RIGOLL, G. (Hrsg.): *Mustererkennung 2008*. Berlin : Springer-Verlag, 2008 (LNCS), Seiten 61–70

[Sobel & Feldman, 1968] SOBEL, I.; FELDMAN, G.: *A 3x3 isotropic gradient operator for image processing* / Stanford Artificial Project. 1968. – Forschungsbericht

[Stanley & Miikkulainen, 2002] STANLEY, K.; MIIKKULAINEN, R.: *Evolving Neural Networks through Augmenting Topologies*. In: *Evolutionary Computation*, 10 (2002), Nr. 2, Seiten 99–127

[Stenzel, 2002] STENZEL, R.: *Steuerungsarchitekturen für autonome mobile Roboter*. Diplomarbeit. RWTH Aachen, Fakultät für Mathematik, Informatik und Naturwissenschaften, 2002

[Stone, 1998] STONE, P.: *Layered Learning in Multi-Agent Systems*. – Dissertation. School of Computer Science, Carnegie Mellon University, December 1998

[Stone et al., 2005] STONE, P.; SUTTON, R.; KUHLMANN, G.: *Reinforcement Learning for RoboCup-Soccer Keepaway*. In: *Adaptive Behavior*, 13 (2005), Nr. 3, Seiten 165–188

[Sutton & Barto, 1998] SUTTON, R.; BARTO, A.: *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts : The MIT Press, 1998

[Sutton et al., 1999] SUTTON, R.; PRECUP, D.; SINGH, S.: *Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning*. In: *Artificial Intelligence*, 112 (1999), Seiten 181–211

[Taylor et al., 2008] TAYLOR, M.; KUHLMANN, G.; STONE, P.: *Autonomous transfer for reinforcement learning*. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, 2008

[Thrun et al., 2005] THRUN, S.; BURGARD, W.; FOX, D.: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. Cambridge, Massachusetts : The MIT Press, 2005

[Thrun et al., 2000] THRUN, S.; FOX, D.; BURGARD, W.; DELLAERT, F.: *Robust Monte Carlo Localization for Mobile Robots*. In: *Artificial Intelligence*, 128 (2000), Nr. 1-2, Seiten 99–141

[Tsitsiklis & Roy, 1996] TSITSIKLIS, J.; ROY, B.: *Analysis of Temporal-Diffference Learning with Function Approximation*. In: *NIPS*, 1996, Seiten 1075–1081

[Vapnik, 1998] VAPNIK, V.: *Statistical Learning Theory*. New-York : Wiley-Interscience, 1998

[Watkins, 1989] WATKINS, C.: *Learning from Delayed Rewards*. – Dissertation. King's College, Cambridge University, May 1989

[Wikipedia, 2011a] WIKIPEDIA: *Anpassungsfähigkeit*. 2011. – URL `http://de.wikipedia.org/wiki/Anpassungsf\%C3\%A4higkeit`

[Wikipedia, 2011b] WIKIPEDIA: *Autonomer mobiler Roboter, Hybride Architekturen*. 2011. – URL `http://de.wikipedia.org/wiki/Autonomer\_mobiler\_Roboter`

[Wikipedia, 2011c] WIKIPEDIA: *Lernfähigkeit*. 2011. – URL `http://de.wikipedia.org/wiki/Lernf\%C3\%A4higkeit`

[Wikipedia, 2011d] WIKIPEDIA: *Semi-Markow-Prozess*. 2011. – URL `http://de.wikipedia.org/wiki/Semi-Markow-Prozess`

[Wolter, 2007] WOLTER, A.: *Reinforcement Learning in der Roboternavigation*. – Diplomarbeit. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, September 2007

[Zell, 2003] ZELL, A.: *Simulation neuronaler Netze*. Oldenbourg Wissenschaftsverlag, 2003

[Zhigun, 2011] ZHIGUN, M.: *Ranging sensor system for indoor mobile robot*. – Master Thesis. Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Germany, 2011